Measuring and Modelling RPC Performance in OSF DCE

A Thesis Submitted to the College of Graduate Studies and Research
in Partial Fulfillment of the Requirements For the Degree of Master of Science in the Department of Computer Science

University of Saskatchewan
Saskatoon, Saskatchewan

by

Ying Sun

November 1997

© Copyright Ying Sun, 1997. All rights reserved.
Permission To Use

In presenting this thesis in partial fulfillment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
University of Saskatchewan
Saskatoon, Saskatchewan, Canada
S7N 5A9
Abstract

Middleware is an enabling layer that enables developers to build distributed applications on heterogeneous computer systems that are provided by different vendors. Middleware provides to the distributed applications a group of distributed services that mask the details of different operating systems and networks.

The Open Software Foundation's Distributed Computing Environment (DCE) is the current de facto standard for middleware. DCE is based on the popular client/server model. The communication – the core service of DCE – is performed by the Remote Procedure Call (RPC) facility. RPC performance is of vital importance to the performance of both DCE applications and the DCE system, because RPC forms the basis of communication for DCE and DCE applications.

The RPC communication of DCE is performed by the RPC Runtime, which works according to the RPC protocols. RPC data is fragmented/reassembled by the RPC protocols and the lower layer network protocols. This adds overhead at both the client/server processor and the network, affects the RPC performance, and, with other operations introduced by RPC protocol, make the RPC-caused network traffic bears certain characteristics. Understanding the characteristics of RPC-caused network traffic is useful in understanding issues relating to DCE performance.

Many factors affect DCE RPC performance. Among them, network utilisation and underlying transport protocols are important because RPC relies on the network for communication. Another important factor is interoperation, the need to operate across different platforms, because DCE is designed to help the distribution of applications on different platforms and operating systems.

This research deals with the performance of DCE RPC. The focus is on the measurement and modelling of the packet arrival process of RPC-caused network traffic in the DCE environment, and the measurement of the latency and capacity of DCE RPC when running synthetic DCE applications, as a function of network utilisation, underlying transport protocols, and interoperation of platforms/OSs. The purpose is to provide a better understanding of RPC and its performance issues.
Acknowledgements

A special acknowledgment to my supervisor, Dr. Richard Bunt, for his constructive supervision and financial support on my research. And more than that, for all his assistance and guidance started even before I arrived Canada. Thanks to my committee members, Dr. Derek Eager, Dr. Eric Neufeld, and my external examiner, Dr. Ron Bolton (Electrical Engineering) for their helpful suggestions on my work. I would like to acknowledge IBM Canada, Centre for Advanced Studies, for providing me the opportunity to study in their Lab, and supporting me financially during my research there.

I would like to thank my fellow graduate students in the Department, for their help, and for the good times we spent together during my study. Also, I would like to thank the members of the DISCUS research group, and in particular, its technical member, Greg Oster, who worked together with me in the DCE project, for his assistance and useful advises. Thanks are also extended to the secretary/management members of the Department, Mrs. Gail Walker and Mrs. Maureen Desjardins, who are always ready to help when needed. Thanks to my host family, Mrs. Barb Jinkense and her family, for their help and kindness.

A very special thanks is reserved to my parents, Naiyuan Sun and Chenli Tong, and my sister, Hui Sun. Although they are tens of thousands miles away, I can always feel their encouragement and support.

Finally, but by no means unimportant, I would like to thank my husband, Ruifeng Wang. It’s more than an acknowledgment, because we share everything – either hardship or success, in our life.
# Table Of Contents

Permission To Use ........................................... i  
Abstract ................................................................ ii  
Acknowledgements .............................................. iii  
Table Of Contents ............................................... iv  
List of Tables ...................................................... vii  
List of Figures ..................................................... ix  

1 Introduction ................................................... 1  

2 Distributed Computing and DCE ................................. 4  
2.1 Introduction .................................................. 4  
2.2 The Origins of DCE ........................................... 4  
2.2.1 The Distribution of Computing Services ............... 4  
2.2.2 Heterogeneity and Middleware .......................... 6  
2.2.3 The History of DCE ....................................... 8  
2.3 DCE Organisation and Services ............................ 9  
2.3.1 The Organisation of DCE ............................... 9  
2.3.2 The DCE Services ....................................... 11  
2.4 Distributed Applications in DCE .......................... 15  
2.4.1 Application Development in DCE ...................... 15  
2.4.2 Running a Distributed Application in DCE .......... 21  

iv
# 2.4.3 The Workload in a Cell

2.5 Summary

## 3 DCE RPC Facility and Protocols

3.1 Introduction
3.2 Background of DCE RPC
3.3 DCE RPC at Runtime
3.4 The DCE RPC Protocols
   3.4.1 Protocol Overview
   3.4.2 Fragmentation of the DCE RPC Data
   3.4.3 Flow Control
3.5 Performance Aspects
3.6 Summary

## 4 Related Studies of DCE and DCE RPC Performance

4.1 Introduction
4.2 Studies of DCE Performance
4.3 Studies of DCE RPC Performance
4.4 Factors Affecting DCE RPC Performance
4.5 Summary

## 5 The Experimental Work

5.1 Introduction
5.2 The Purpose of the Experimental Work
5.3 Measurement Methodology
   5.3.1 The DCE Cell
   5.3.2 The DCE Application
   5.3.3 The Workload
   5.3.4 Tools
   5.3.5 Levels of Factors in the Measurement Environment
   5.3.6 Trace Collection

v
List of Tables

2.1 The Protocol Sequences ............................................. 17

5.1 Upper Bounds of PDU Sizes (in bytes) in Different RPC Implementations ............................................. 58

5.2 Client/Server Pair Combinations and Names .................. 62

6.1 Mean Interarrival Times (msec) Under Light Network Load for Connection-oriented RPC ............................................. 76

6.2 Variances of Inter-arrival Times Under Light Network Load for Connection-oriented RPC ............................................. 76

6.3 CoVs of Interarrival Times Under Light Network Load for Connection-oriented RPC ............................................. 77

6.4 Interarrival Time Statistics for OS/2-to-OS/2 Pair Connection-oriented RPC Under Heavy Network Load ............................................. 80

6.5 Interarrival Time Statistics for Digital UNIX-to-Digital UNIX Pair Connection-oriented RPC Under Regular Network Load ............................................. 81

6.6 Interarrival Time Statistics for Digital UNIX-to-OS/2 Pair Connection-oriented RPC Under Regular Network Load ............................................. 82

6.7 Mean Interarrival Times (msec) Under Light Network Load for Connectionless RPC ............................................. 86

6.8 Variance of Interarrival Times Under Light Network Load for Connectionless RPC ............................................. 86

6.9 CoV of Interarrival Times Under Light Network Load for Connectionless RPC ............................................. 87
6.10 Statistics of Inter-arrival Times of PDU/IP With Regular Network Load for Connectionless RPC ....................... 88

6.11 Comparison of Connectionless and Connection-oriented RPCs on Interarrival Time Statistics (OS/2-to-OS/2 Pair) .................... 92

7.1 Parameters Used in the O.L.E. Model (from Measurement of Connection-oriented RPC, the OS/2-to-OS/2 Pair) ....................... 116

7.2 Parameters Used in the O.L.H. Model (from Measurement of Connection-oriented RPC, the OS/2-to-OS/2 Pair) ....................... 116

7.3 The Two-level Models ........................................... 118

7.4 Parameter Values (Measured Interarrival Time Statistics) Used in the T.L.E.E. Model ............................................. 119

7.5 Parameter Values (Measured Interarrival Time Statistics) Used in the T.L.H.E. Model and T.L.H.H Model - Heavy Network Load Only . 120

7.6 Comparing Modelled and Measured Values for Inter-IP Process Under Light Network Load over Connection-oriented RPC, the OS/2-to-OS/2 Pair .................................................. 120

7.7 Comparing Modelled and Measured Values for Inter-IP Process Under Heavy Network Load over Connection-oriented RPC, the OS/2-to-OS/2 Pair .................................................. 121
List of Figures

2.1 The Role of Middleware ........................................ 7
2.2 Organisation of DCE Services ................................. 10
2.3 DCE Application Structure ................................... 18
2.4 Data and Information Transfer in Running DCE Applications ........................................ 23
2.5 Communication Between the Stubs ............................ 25
2.6 Client/Server Structure and Workload in DCE ................. 26
3.1 Processing of RPCs for Ethernet Transport ..................... 32
3.2 The Structure of an RPC PDU ................................ 33
3.3 The Fragmentation Layers .................................... 37
3.4 Transmitting PDUs Under Slow Start (from [30]) .............. 39
4.1 Round Trip Time as a Function of RPC Data Size (Results from [23]) 49
4.2 Comparison of RTT for Different RPC Protocols (Results from [11]) . 50
5.1 The Experimental Environment ................................. 56
5.2 Table Display of the Traffic Generator ........................ 61
5.3 Definitions of Interarrival Times and RTT ...................... 67
6.1 Interarrival Time Distributions of the OS/2-to-OS/2 Pair Under Different Network Loads for Connection-oriented RPC .............. 79
6.2 Interarrival Time Distributions of the Digital UNIX-to-Digital UNIX Pair Under Different Network Loads for Connection-oriented RPC . 100
6.3 Interarrival Time Distributions of the Digital UNIX-to-OS/2 Pair Under Different Network Loads for Connection-oriented RPC ........ 101
6.4 Mean Value and Standard Deviation of Inter-arrival Times for the OS/2-to-OS/2 Pair Under Different Network Loads for Connection-oriented RPC ............................................. 102
6.5 Interarrival Times and Their Standard Deviations for Different Transmission Units under Light Network Load for Connection-oriented RPC 103
6.6 Interarrival Times and Their Standard Deviations for Different Transmission Units under Light Network Load for Connectionless RPC, with the OS/2-to-OS/2 Pair .................................................. 103
6.7 Interarrival Time Distributions of the OS/2-to-OS/2 Pair under Light Network Load for Connectionless RPC ................................................................. 104
6.8 Interarrival Time Distributions for PDU and IP of the Digital UNIX-to-Digital UNIX Pair under Different Network Loads for Connectionless RPC ......................................................... 105
6.9 Interarrival Time Distributions for PDU and IP of the Digital UNIX-to-OS/2 Pair under Different Network Loads for Connectionless RPC 105
6.10 RPC Round-trip Time Measurement under Different Network Loads for Connection-oriented RPC ................................................................. 106
6.11 RPC Throughput Measurement under Different Network Loads for Connection-oriented RPC ................................................................. 107
6.12 The Effect of Different Network Loads on RPC RTT for Connection-oriented RPC ................................................................. 107
6.13 RPC Round-trip Time Measurement under Light Network Load for Connectionless RPC on the OS/2-to-OS/2 Pair .................................................. 108
6.15 RPC Throughput Measurement under Different Network Loads for Connectionless RPC ................................................................. 109
6.16 RPC Round-trip Time Measurement under Different RPC Protocols under Light Network Load ................................................................. 109
7.1 The Two-level Models in the Experiment ........................................ 119

7.2 The Modelled and Measured Inter-IP Time Distribution Under light
Network Load for Connection-oriented RPC, the OS/2-to-OS/2 Pair . 121

7.3 The Modelled and Measured Inter-IP Time Distribution Under Heavy
Network Load for Connection-oriented RPC, the OS/2-to-OS/2 Pair . 123
Chapter 1

Introduction

Distributed applications enable end users to share resources and services across interconnected computer systems. In developing distributed applications, what challenges the ingenuity of designers is the fact that the networked computer systems are heterogeneous.

Heterogeneity in a computer system means that the system contains different hardware platforms, running different operating systems and, sometimes, connected with different networks. The problems of developing distributed applications and managing the execution of those applications in such environments are of considerable interest to the industry, and present challenges to the researchers.

Middleware [5] has been introduced to simplify the development and management of distributed applications in heterogeneous environments. Middleware is a layer of software services that is built upon the different operating systems and networks to support the distribution of user-level services. It takes care of the details of the underlying operating system(s) and network(s), and provides distributed services to the upper layer. The application developers have only to use the distributed services provided by the middleware to develop distributed applications. The Open Software Foundation’s (OSF) Distributed Computing Environment (DCE) [32] is the current de facto standard for middleware.
DCE sits on top of existing operating systems and provides distributed services and application program interfaces to the applications based on the client/server model. The performance issues of DCE are related to the performance issues of distributed systems, of middleware, and of the applications running on top of middleware. Therefore, both industry and researchers have considerable interest in the performance of DCE.

The clients and servers of both DCE and DCE applications communicate with each other using Remote Procedure Call (RPC) [8]. Because RPC is central to DCE client/server communications, DCE RPC becomes a very important entity in the distribution of DCE services and DCE applications, and DCE RPC performance is of vital importance to both DCE application performance and DCE system performance. The study of DCE RPC performance requires a thorough understanding of DCE RPC. Because of the short time that DCE has been around, DCE RPC is not well understood by researchers. Former performance studies of DCE RPC have provided some understanding of the RPC mechanism, the factors that affect RPC performance, and some measurement results that come from homogeneous environments. Other issues, such as network factors, overheads introduced by RPC protocols, and heterogeneity of the environment on which DCE and DCE applications are running have received less attention.

Workload is an important component of any performance study, and DCE is no exception. Because the network supports the communication, DCE RPC performance is highly influenced by network performance. An accurate DCE RPC workload model helps in the performance study of the network, and also provides more understanding of DCE RPC, which helps in making design decisions to improve the performance of DCE RPC itself. Based on the measurement of DCE RPC workload characteristics, useful DCE RPC workload models can be developed.

This research is aimed at providing a better understanding of DCE RPC. To
achieve this, DCE RPC is studied from three perspectives, with the following goals:

1. To study the design decisions and operational semantics of DCE RPC, including the RPC protocols.
2. To measure RPC performance, and determine the factors that affect it in the DCE environment.
3. To measure and model the DCE RPC generated workload on the network. The characteristics of this workload are affected by both the internal design decisions of DCE RPC and external factors such as network load.

This research is based on experimental work done in a heterogeneous DCE environment in the DISCUS Lab in the Department of Computer Science at the University of Saskatchewan.

In this thesis, Chapter 2 provides some background on DCE. It describes the architecture of DCE and services provided by DCE. It also describes the developing and running of DCE applications. Chapter 3 describes the RPC facility and the RPC protocols, including some of the details of the RPC protocols that are relevant to this research. Related previous performance studies on DCE and DCE RPC are reviewed in Chapter 4. Chapter 5 gives a description of the experimental testbed and methodology, and the specific issues to be investigated in this thesis. In Chapter 6, the results of the experimental measurements are presented and discussed. Work on modelling the packet arrival process of DCE RPC-caused network traffic are presented and discussed in Chapter 7. Finally, in Chapter 8, the thesis is summarized and directions for future work are provided.
Chapter 2

Distributed Computing and DCE

2.1 Introduction

The demand from end users to share resources and services across multi-vendor computer systems led to the need to develop distributed applications in heterogeneous environments. DCE has emerged as middleware to assist in this process.

This chapter describes the goals, history and structure of DCE, the services it provides, and how it supports the development and execution of distributed applications.

2.2 The Origins of DCE

2.2.1 The Distribution of Computing Services

Sharing of resources and services has been a big issue in the evolution of computer systems. CPU's, peripheral devices and data can all be considered as resources. The computer system, with the support of its hardware and software resources, provides various kinds of services. End users access these resources and services through applications.

In the 1970's, resources and services were accessed in a centralized way, i.e. one single computer was one system, many users were served by a centrally managed
mainframe, and sharing with other computers was difficult.

In the 1980's, many computer systems consisted of personal computers and workstations connected together by a network. The resources and services were related to individual computers. Although users could share a certain resource or service, they had to know on which computer the resource or service was located, and use some programming language or system level command to access it. The application developers had to "hard code" the application software to the operating systems and the network in order to provide sharing to the end user.

The demand to share resources and services across the networked computers by end users led to the demand for distributed applications in the late 1980's. The distribution of the applications is based on the distribution of the computing services. The distributed services can be provided by the operating system, because operating systems control the bare resources of a computing system. Also, there can be a layer of distributed services built on top of the operating systems and networks. However they are provided, the distributed services that are required by the distributed applications must include at least the following:

- a communication service
- a naming service
- resource management

The related technologies have been studied for years. Many vendors of computer systems have developed their own software packages of distributed services. These services are always based on the operating system and network architecture used by a vendor or a group of vendors. For example, distributed file services are provided by NFS (Network File System) in UNIX systems. When writing distributed applications for the computer systems that support the same distributed service package, the software developers have only to make function calls to the services.
The distributed services provided this way have simplified distributed application development in homogeneous computer systems.

2.2.2 Heterogeneity and Middleware

In reality, the "system" seen by end users consists of different hardware platforms from different vendors running different operating systems (OS's). A current demand from end users is for distributed applications across multi-vendor systems. This presents a challenge to the application developers because the distributed services that are available on the systems from different vendors are often distinct from each other. The developers must manage all the system and communication details without any assistance from the distributed services.

To help solve the distribution problems in heterogeneous environments, there emerged the notion of middleware service [5]. A middleware service is a general-purpose service that sits between platforms and applications. The "platform" here includes both the hardware platform and the operating system platform. Middleware services appear on systems that are provided by different vendors, in the place of the previously vendor-related distributed services, and mask the heterogeneity of the platforms to the distributed applications. Integrating a set of middleware services to make them work as a coherent system adds value to the set of middleware services and results in middleware (also called midware). Application developers access specific middleware services via the Application Programming Interface (API) that is provided by the middleware. Figure 2.1 shows the difference between middleware and previous vendor-related distributed service packages, and helps in understanding the middleware idea. In the diagram, there are two kinds of systems; each has four computers, running different operating systems A, B and C, connected by a network. In the first system (the top one on the diagram), application 1 is distributed between two computers (computer 1 and computer 2) that both run OS A, directly supported by distributed services-I which is related to OS A. The other two com-
puters run different OS's, OS B and OS C. Although OS B and OS C both support their own distributed services, distributed services-II and distributed services-III, respectively, the distributed services are disparate. If the developers want to develop an application (application 2) that is distributed between computers 3 and 4, they have to manage the details of OS B and OS C, without any assistance from distributed services-II and distributed services-III. In the other system, (the bottom one on the diagram), a layer of middleware services replaced the distinct distributed services for different platforms. Any application can be distributed among the four computers in the system by simply calling the middleware services that are provided across platforms.

Figure 2.1: The Role of Middleware

Middleware is a solution to the problem of creating distributed applications for heterogeneous computer systems, and provides the application developers with an infrastructure to create, deploy and manage the distributed applications more easily, more quickly, and more cheaply. It is suggested that well-designed and well-implemented middleware should be as coherent, easy to share, expandable, and scalable, as possible. As middleware, the Open Software Foundation’s Distributed
Computing Environment (OSF DCE) is becoming increasingly well-known.

2.2.3 The History of DCE

DCE came from industry. Founded in 1988 by a group of computer companies, the Open Software Foundation (OSF) aimed at developing a new version of UNIX, called OSF/1, to compete with the UNIX versions from UI (UNIX International). The OSF consortium consists of dozens of major computer vendors, including IBM, DEC and HP. These vendors needed to build their distributed application products on top of OSF/1 and other UNIX (or non-UNIX) systems. In response to OSF's "Request for Technology", many companies and research agencies made their offer to provide parts of DCE. OSF chose among them. The original sponsors for the chosen technologies are: HP/Apollo and DEC for RPC, Siemens AG for Name Service, HP/Apollo for Security Service, DEC for Time Service and Multi-threading, and Transarc for the Distributed File System. OSF developed the chosen technical components further, and integrated them into the DCE software package. OSF DCE was first introduced in 1990.

At present, OSF DCE is the foundation of the middleware products of companies such as IBM, Hewlett Packard, Digital Equipment, etc. DCE provides distributed services and distributed application development support to users in a variety of industries including the financial industry, the telecommunications industry, and manufacturing. The DCE user group also consists of government agencies, software companies and users in other areas.

The next section describes the architecture of DCE and the services provided.
2.3 DCE Organisation and Services

2.3.1 The Organisation of DCE

DCE provides a group of distributed services to the users on the machines connected by the network. The organisation of DCE services refers to the relationships between the DCE services that reside on the machines.

There are five fundamental services provided by DCE (also called DCE components). These are:

1. Distributed Directory Service (including Cell Directory Service (CDS) and Global Directory Service (GDS));
2. Distributed Security Service (DSS);
3. Distributed Time Service (DTS);
4. Threads Service;
5. Remote Procedure Call (RPC) Service;

The DCE components appear on every machine that runs DCE, and work in integrated fashion with recursive requests to each other, as shown in Figure 2.2. The Threads and RPC services are built on top of the operating system and network, and provide support for other services. Some recursive requests among the components are shown with double headed arrow lines. The distributed file service is supported by the DCE components. It can be provided by DCE's Distributed File Service (DFS) as well as other distributed file systems. The DCE components and distributed file service are called the DCE services. Application Programming Interfaces (API) of the DCE services are offered as C libraries, which are convenient for use by the application developers.
A group of machines that typically have a common purpose, share common DCE services, and are grouped for particular administrative purposes, is called a cell. This is the fundamental DCE organisational unit. A cell is formed by linking hosts and other resources with networks (either LAN or WAN), for the purpose of administration, security and usage. Most of the DCE services are provided within a cell. Distribution across cells (via global services) is also possible but it is more expensive. DCE provides the services based on the client/server model. The client/server model defines the principles of distributed computing as follows: any process that requires service from the system is called a client, any process that can provide service(s) to the system is called a server. The host on which the client process runs is called a client (machine), while the host on which the server process runs is called a server (machine). In DCE, the hosts on which the CDS, DSS or DTS server processes run are called the DCE servers. There must be at least one CDS server, one Security server, and one DTS server in a cell, to provide the necessary DCE
service. The DCE server can be any suitable host within the cell. Other hosts run the client processes of the DCE services, and are called the DCE clients. A host on which one or more DCE server processes reside may, at the same time, have client processes to other DCE services reside on it, and so it can be a DCE server and a DCE client simultaneously.

2.3.2 The DCE Services

Directory service provides name service. This is important to any distributed system, including DCE. It is used in finding resources in the system, and helps the application clients to find the servers. Both cell directory service (CDS) and global directory service (GDS) are parts of DCE distributed directory service. CDS controls the names within a cell. GDS locates resources in foreign cells. They support both CCITT X.500 standard (XDS) and Domain Name Service (DNS) in the Internet.

Every resource in a cell has a unique name and attributes. Logically, they form "directory entries" and are organised hierarchically. Physically, the directory entry groups and their replications are stored in a database called the clearing house that resides on the CDS server. Users access clearing houses to find information about the resource (such as its address and attributes). These resources are then accessed through other DCE services. Every cell has a CDS server and one or more CDS clients. Because data sharing happens in a cell and among cells, technologies such as caching, replication and consistency management are used to improve the performance of directory service. GDS is more complicated than CDS. It is used to look up a name outside a DCE cell. It is a higher level directory service that connects the cells (i.e. that connects the CDS in each cell). To be connected to the outside world using GDS, there is a Global Directory Agent (GDA) in each cell. The GDA is connected to the CDS, takes a name that can not be found in the local cell and finds its location by using the GDS.
The Distributed Directory Service also provides file systems with name service. Because CDS stores information such as where the file resides, it can be integrated with DCE Distributed File Service (DFS) or other distributed file systems to provide a global name space, while the real files are managed by the DFS or other file systems. Whether or not a DCE installation includes DFS, the integration of file systems and CDS is possible. Application programmers access the DCE Directory Service via the X/Open Directory Service (XDS) application programming interface. By calling XDS library routines, programmers can access both the CDS and the GDS.

The DCE security service provides authentication and authorization services by means of user registry, login control to DCE hosts, and access control to the resources. There must be one security server in every DCE cell. The security server consists of three server processes: registry server, authentication server and authorization (privilege) server, all with access to the security database. The authentication service verifies the identities of the client and server processes, and the authorization service checks the privilege of users to access a certain resource. The application programming interface (API) of the DCE security service is Authenticated RPC. By programming with the Authenticated RPCs, developers can make the application work on different security levels when communicating. Higher security levels require more complicated authentication and data encryption on each RPC call, and therefore more workload is added to the system and the application runs more slowly.

The purpose of the Distributed Time Service (DTS) is to keep the clocks on different machines within the cell synchronized and correct (i.e. reasonably close to the Coordinated Universal Time). In a cell, at least three DTS servers are recommended. Other nodes of the cell run a DTS client called a time clerk. The time clerks periodically (depending on the hardware clock of the host) send queries to
the time server for the correct time, making the local clock keep pace with the cell
time. The DTS servers also keep sending queries to one another. The one design-
nated as “master” calculates the time intervals sent from other time servers, finds
the synchronized time, and broadcasts the new time back to them. A similar method
is also used to synchronise clocks among cells. The application programmers can
access DTS either by retrieving the time from the system, or by using DTS API
routines.

Distributed File Service (DFS) is another important DCE service. DFS provides
data sharing services to the users. Since it is supported by the fundamental DCE
services (DCE components), some view it as a DCE application rather than a DCE
component. User-developed applications can use DFS and benefit from the security,
load balancing, scalability, replication and other mechanisms. Users can also choose
not to use DFS and use their own file system while being supported by other DCE
services.

The DCE Threads Service supports multi-threading in DCE and DCE appli-
cations. The term “thread” refers to the thread of control in the execution of a
program. Multi-threading allows more than one thread of control to be active dur-
ing program execution. In DCE, the Threads Service is provided as a user-space
library. A host system can either install the DCE threads software package (if it
doesn’t have a kernel threads facility in its operating system) or set DCE to use
the threads package that comes with the operating system kernel. Whatever kind of
threads package it installs, there are two ways to use this mechanism – multi-threads
can be handled internally by the RPC service, or programmers can create threads
in their application programs. DCE also provides ways for its threads to interact
with other software.

It can be argued that the most important component of DCE is the Remote
Procedure Call (RPC) Service, because the clients and servers of DCE and DCE
applications communicate with each other using RPC. RPC is an extension of the familiar procedure call mechanism. With the RPC mechanism, a call to a procedure on a remote machine looks as if it were local. The communication details are handled by the RPC facility.

The DCE RPC facility has three major components: the Interface Definition Language (IDL) and IDL compiler, the Universal Unique Identifier (UUID) generator, and the RPC Runtime. The application programmers use the RPC facility by generating a unique interface for a certain application program using the UUID generator, defining the interface using IDL, and making RPC calls directly in application programs. The execution of the applications is supported by the RPC Runtime.

Application developers use IDL, a C-like language, to describe the client/server interfaces. The result of compiling an IDL file is referred to as stub code. The caller procedure on the client side and the callee procedure on the server side both communicate with their own stub code, making the remote procedure behave as if it were local. The stub codes call the RPC Runtime routines to manage the client/server communication. The RPC Runtime supports two RPC protocols to handle the communication details: RPC over connection-oriented transport (eg. TCP/IP), the connection-oriented RPC protocol, and RPC over connectionless transport (eg. UDP/IP), the connectionless RPC protocol. The details of writing distributed applications with the DCE RPC facility will be described in a later section.

In the literature, the terms "RPC Facility", "RPC call", "RPC protocols" and "RPC Runtime" are all referred to as "RPC". To alleviate confusion these terms are distinguished in this research. The "RPC Facility" refers to the three components that contribute to the RPC mechanism: IDL and the IDL compiler, the UUID generator, and the RPC Runtime service. An "RPC call" refers to one remote procedure call from/to the DCE/application client/server. The "RPC protocols"
refer to the connectionless and connection-oriented RPC protocols, and the "RPC Runtime" refers to the RPC Runtime library and runtime service.

2.4 Distributed Applications in DCE

Application developers use the RPC Facility to create the DCE applications. The developer defines the interface and writes the client, server, and server side remote procedure code. The RPC Facility generates the communication part of the application – the client and server stub, – and manages the communications.

The running of a DCE application consists of two parts: the client process finding the server host/process, and the client requesting the service from the server by means of remote procedure calls. The first part is called binding. Binding is done only when the application client begins to run. The client/server communication is performed automatically by the stubs generated by the IDL compiler.

2.4.1 Application Development in DCE

DCE provides an infrastructure to develop distributed applications in heterogeneous computer systems. Without knowing the details of the underlying operating systems and networks, the application developers have only to do the following four pieces of work: defining the data structure used in the data communication (usually an RPC) which is the interface between the client and the server; writing the server source code to initialise the server; writing the client source code to make the remote procedure calls; and writing the procedure code that does the real work the client requires on the server side. Compiling the interface definition file (".idl" file) results in one public header file, one client stub, and one server stub. It is the stubs that manage the sending and receiving of the RPCs, and the format of the transferred data.
Because the application is identified by the interface, a Universal Unique IDentifier (UUID) is associated with each interface. This UUID is obtained by a UUID generator, and is included in the .idl file. The client stub takes the responsibility of converting the client data to a standard format for transmission. When the data reaches the server, it is converted from the transmitting format to the server-acceptable format by the server stub. The result data is converted by the server stub to the standard transmitting format and returned from the server side to the client side. It is then converted to the client-acceptable format by the client stub. Client and server stubs communicate with application programs, and communicate with the RPC Runtime too. The RPC Runtime then transfers the data between the application client and server host, basing the transfer on the chosen RPC protocols.

The server code of a DCE application registers the interface with the RPC Runtime, creates the server binding information and starts listening to the RPCs. The created binding information consists of three parts: the protocol sequence, the name and address of the server host, and the server process ID (endpoint). This information is either stored in the name service databases/server endpoint map or printed out by the server as an alphabetic string (i.e. string binding). When using string binding, manual operations by the user are required. The server needs to tell the client, before the client begins to run, how the binding information is provided, so that the client can obtain it properly. The server is able to choose which RPC protocol to use by listening on a certain RPC protocol sequence. The current RPC protocols are connectionless and connection-oriented RPC protocols, and the respective RPC protocol sequences are described by the format in Table 2.1. The RPC protocols "ncacn" and "ncadg" mean "Network Computing Architecture ConNection oriented protocol" and "Network Computing Architecture DataGram protocol", respectively.

On the client side, there are three steps to the client process:

1. Obtaining the binding information and setting the binding handle (a data
Table 2.1: The Protocol Sequences

<table>
<thead>
<tr>
<th></th>
<th>Connection-oriented</th>
<th>Connectionless</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPC protocol</td>
<td>ncacn</td>
<td>ncadg</td>
</tr>
<tr>
<td>Network address</td>
<td>Internetwork</td>
<td>Internetwork</td>
</tr>
<tr>
<td>format Protocol</td>
<td>Protocol (IP)</td>
<td>Protocol (IP)</td>
</tr>
<tr>
<td>Transport</td>
<td>TCP</td>
<td>UDP</td>
</tr>
<tr>
<td>protocol</td>
<td>Protocol sequence</td>
<td>Protocol sequence</td>
</tr>
<tr>
<td></td>
<td>ncacn.ip.tcp</td>
<td>ncadg.ip udp</td>
</tr>
</tbody>
</table>

structure for binding-related information);

2. Calling the remote procedure;

3. Passing the binding handle to RPC Runtime.

These tasks are done through cooperation of the application client code and the client stub. The application developers are able to choose how much binding they will be responsible for, by choosing the binding method before writing the interfaces and the code. Three binding methods are provided in DCE: automatic binding, implicit binding and explicit binding.

In automatic binding, the client stub does both step 1 and step 3. The application has only to make the RPC call. By the time the first RPC call is made, the client is bound to the server. In implicit binding, the binding handle is defined as a global variable by the client stub. The application client does step 1 and step 2, and passes the binding information to the stub by assigning the global variable. In explicit binding, the application also has to do step 1 and step 2, but the binding handle is passed to the client stub as a parameter of the RPC call. In both implicit and explicit binding, the client stub is responsible only for passing the binding handle to RPC Runtime. In choosing the binding method, the application developer weighs several factors: automatic binding is simple, but it takes far more time to finish the first RPC; explicit binding is flexible, but has an extra parameter in the
RPC call; implicit binding uses the global variable. Before writing the application, the developer has to decide which binding method to use.

The real work required by the client is done by the procedure code which resides on the server side. It is the actual application service program. This code is often a program, consisting of the public header file generated by the IDL compiler.

The object files of the server, the server stub and the procedure are linked together to form the application server. The object files of client and client stub are linked to form the application client. Figure 2.3 shows the structure of DCE applications. In the diagram, the developer writes the interface definition and the C programs of the client, the server, and the remote procedure. The RPC Facility uses the interface definition to generate the header file and the client and server stubs.
Linking the object files of the server and the server stub with the RPC Runtime library results in the executable application server. Linking the object files of the client and client stub with the RPC Runtime results in the executable application client.

The following example shows what a DCE application looks like. "Rshow" is a simple application in which the client uses a remote procedure call to print a string of characters on the server, and gets back a reply string from the server. The application is made up of four pieces: "rshow.idl" - the interface definition, "rshow.sr.c" - server code, "rshow.cl.c" - client code, and "rshow.mg.c" - the code which contains the real remote procedure that is initiated by the server.

The interface definition file "rshow.idl" listed below defines the remote procedure (named showmsg()) and its parameters. An explicit binding method is used (the first parameter of the remote procedure).

```c
#else The interface definition. File name: rshow.idl. */
[
  uuid(007704c0-4515-113e-aacc-00a0242fa264),
  version(1.0)
]
/* In the above two lines, a UUID generated by the UUID generator is referred to by the interface, followed by a version number */
interface rshow
{
  void showmsg( [in] handle_t hdl,
                  /* The binding handle used in explicit binding */
                  [in, string] char mess[128],
                  /* Input data, message from the client */
                  [out, string] char reply[128]);
  /* Output data, the reply message sent to the client */
}
Compiling the "rshow.idl" file results in the public header file "rshow.h", and the stubs: "rshow_client.stub.o" and "rshow_server.stub.o". The header file is included in every source file that is generated thereafter. The stubs manage the client/server communication (passing of the parameters, etc.).

The parts that concern the remote procedure in the client and server source code are shown as follows. The client only calls the remote procedure, the server listens and captures the request, and then calls the remote procedure which prints the string.

/* Client code where the remote procedure showmsg() is called. */
/* These statements are found in the client source code file rshow_cl.c. */

(binding)

... showmsg(handle_of_rshow, "Are you there, server?", answer);
/* The binding handle is taken from the binding procedure previously. The client is calling the remote procedure with an input string of "Are you there, server?", and an output string variable called "answer" */

... The server only initializes itself, and listens:

/* Server code where the remote procedure request is received */
/* The code that does the following work is found in server source */
/* code file rshow_sr.c. */

( server initializing )

... rpc_server_listen (MAX_CON_CALL_PROT, &status );
/* MAX_CON_CALL_PROT is the maximum number of ports that the server */
/* is listening, status contains the error message */
When the server captures the client request on one of the ports it is listening, it invokes the corresponding remote procedure “showmsg()” to do the real job. The code of showmsg() is found in the file “rshow_mg.c”, which contains all the remote procedures defined in the interface definition file for this particular application.

/* The following are found in the file rshow_mg.c that contains */
/* all the remote procedures for this application */

( Other remote procedures, if any. The order doesn’t matter. )

```c
void showmsg(handle_t hdl,
            idl_char *clt_greeting,
            idl_char *my_reply)
{
    printf("Greetings from the client: %s\n", clt_greeting);
    /* The remote procedure prints out the message from the client
       on the server screen ("Are you there, server?") */

    strncpy(my_reply, "Yes, Client!", 128);
    /* Returns the server answer message to the output string
       variable, in order that it can be bring back to the client */
}
```

2.4.2 Running a Distributed Application in DCE

When it runs, a DCE application calls for the services provided by DCE. Studying the process of application execution helps understand how DCE supports distributed
applications. There are three important processes involved in the execution of a DCE application: server registration, binding, and calling of the remote procedures.

To run a DCE application, the first step is to start the server. When started, the server registers itself with RPC Runtime, and creates the binding information. Whether or not it uses the CDS service and the endpoint map to register the server host and server process is to be decided by the application developer. After the server initiation, the server process begins to listen to the client request on certain network ports.

The application client begins its activity with the binding process, which finds the server. The server information can be passed to the client either by an alphabetic string containing the protocol sequence, the server host network address, and the server endpoint, or by accessing a CDS database on the CDS server and an endpoint database called endpoint map on the application server host. The application developer decides how to pass the binding information from the server to the client.

After the client is bound to the server, the client begins to make the remote procedure calls. Every time the application client calls a remote procedure, the client and server stubs begin to communicate with each other, basing the communication on the RPC protocols. Figure 2.4 shows the data and information transfer during the running of a DCE application.

The important steps that the client stub follows are:

- Initiating the client.
- Obtaining the call handle.
- Marshalling and copying the data to call packets.
- Sending out the call.
• Waiting for the results.

• Getting the results and unmarshalling them, and then passing the results to the application.

• Cleaning up.

In these steps, the call handle is used to hold the state of the RPC call. Marshalling is the process that converts the data into a byte stream format and encodes it for transmission across the network. When the result is received, the data contained in the arriving message undergoes an unmarshalling process to convert the
data from the transmission format to the client-compatible format.

After the RPC call reaches the server side and is caught by the listening server process, the server stub also follows similar steps:

• Initiating the server (different from application-level "server initiation").

• Unmarshalling the received data.

• Passing the data to the remote procedure, and getting the result from it.

• Marshalling the result data.

• Sending the result back to the client side.

• Cleaning up.

Figure 2.5 shows the operation of the client and server stubs.

2.4.3 The Workload in a Cell

As described in a former section, there are DCE clients and DCE servers in a DCE cell. A host in a DCE cell is either a DCE client, or a DCE server, or both. The distributed application running on DCE is also based on the client/server model. Physically, application client procedures reside on either DCE client or DCE server hosts. So too, do application server procedures.

The "DCE workload" is made up of things like queries and replies among the time servers and clients, user logins, the user and resource registry by the administrators, etc. The requests and replies between the application client and server make up the "application workload". Both kinds of workloads are made up of RPC calls. The workload between any two hosts in a DCE cell consists of either one kind of workload mentioned above, or a mixture of both. Figure 2.6 shows two examples of the client/server structure in a DCE cell. In the example marked "A" in the figure,
Figure 2.5: Communication Between the Stubs

the workload between the application client and server is simple, consisting of only the application-caused workload that can be controlled by the application. However, in “B”, the workload caused by the application and by DCE system are mixed together. In this research, when setting up the experimental testbed, a structure similar to A was used, i.e. the application client and server are on separate DCE client hosts, to focus the study on the application-caused RPCs.

2.5 Summary

DCE is middleware to help in developing distributed applications in heterogeneous computing environments. It sits on top of operating systems, and provides distributed services and programming interfaces to the applications, masking the system and network details to the application developers.
DCE consists of five basic components that provide the fundamental distributed services. They are: Distributed Directory Service, Distributed Security Service, Distributed Time Service, Threads Service, and RPC. Distributed File Service is supported by these fundamental distributed services of DCE.

The process of application development in DCE includes defining the interface using IDL, specializing the application by a UUID, and writing client, server and remote procedure code. The details of communication and data representation of different operating systems are handled by the stub codes generated by the IDL compiler. The stub codes communicate with the RPC Runtime, which supports the RPC protocols.

The DCE application client and server processes can reside on either a DCE client host or a DCE server host. The application client finds the server through a binding process. The RPC interactions between the application client and server

Figure 2.6: Client/Server Structure and Workload in DCE
make up the DCE application workload.

This chapter has provided an overview of DCE services, DCE application development and DCE operational semantics. The next chapter deals with the DCE RPC Facility and associated protocols.
Chapter 3

DCE RPC Facility and Protocols

3.1 Introduction

DCE exists to address the problem of heterogeneity of computer systems by facilitating interoperation among different hardware platforms and operating systems. Interoperation is possible only when the heterogeneous systems support the same communication mechanism. In DCE this mechanism is the DCE RPC mechanism, including the communication protocols – the DCE RPC protocols.

The RPC mechanism has been used for a long time in implementing distributed systems. The DCE RPC facility provides the core mechanism in the distribution of services and applications in DCE. The DCE RPC Runtime supports the implementation of the DCE RPC protocol over the underlying transport protocols, manages the communication and data transfer between DCE clients and servers, and thus forms the basis of the DCE RPC facility. The DCE RPC protocol is the core of the DCE RPC Runtime, and the core of communications in DCE.

This chapter discusses the background of RPC technology and some details of the RPC protocols relevant to the RPC performance study.
3.2 Background of DCE RPC

The idea of RPC, remote procedure call, has existed for many years. It has been discussed in the technical literature since at least 1976 [8], and there have been numerous implementations. RPC appears to be a useful technique for providing communication across the network to support the distribution of system services and applications.

RPC is an extension of the well-understood procedure call mechanism, with similar semantics: the caller on one computer issues the RPC call to the callee on another computer, and then suspends execution to wait for the results to come back. The remote procedure is called in the same manner as a local one, with the communication details transparent to the caller and callee.

Some important differences exist between local and remote procedure calls because the caller and callee are usually on different computers, which means that they run in different address spaces and depend on the network to pass the parameters of the remote procedure to the callee and the results back to the caller. Because of complications in sharing address space among computers, it is usually not possible to pass an address as a parameter of a remote procedure. There are also differences in failure semantics. Failures of the remote procedure call can be caused by exceptions in the callee process, failure of the machine (or operating system) on which the callee process is running, or failure in the network. Like local procedure calls, the remote procedures can simply abort, but they can also retry in some cases.

Studies of RPC have shown that there should be at least three parts to help manage the transparent communication of RPC calls in the client/server model: client-side stub (or caller-side stub in a non-client/server model), server-side stub (or callee-side stub), and the RPC Runtime library which supports the RPC protocols.
Before OSF issued its "Request for Technology" for parts of DCE, there were several commercial or experimental RPC packages around. NCS RPC [46], sponsored by HP/Apollo and DEC, was one of the RPC packages that could provide the entire RPC facility, and was chosen by OSF as the core technology to make DCE RPC. In DCE, the RPC Facility is well integrated with other DCE components, supports the communication between the processes, and forms the foundation for the distribution of services and applications in DCE.

3.3 DCE RPC at Runtime

It was discussed in the previous section how the DCE application is developed using two components of the RPC Facility – the UUID generator, and the IDL and its compiler. It has also been mentioned that when the application begins running, both client and server process communicate with their own stubs, making the remote procedure call look as if it were local. The stubs can manage the communication because they are supported by the RPC Runtime.

The RPC Runtime contains a set of standard runtime routines. These routines are called by the stubs of DCE applications and client/server components of DCE services to prepare the data and invoke the RPC calls. For example, when sending data, the stub calls an RPC Runtime routine, so that the Runtime gets the data from the sender process. Then it structures the RPC packets according to the RPC protocols, and passes the RPC packets to the network transport layer for transmission when the stub requires the sending. The RPC and its data are passed from the stubs to the RPC Runtime, to the transport layer of the network, and then to lower layers of the network according to the transport protocols.

Because the stub code is generated automatically, and all the communication details are handled by calling Runtime routines directly, application developers are
freed from considering the details of either system (the word length, byte order, etc.) or network (the transport protocols).

At run time, the application uses the DCE RPC Facility by:

1. Calling the Runtime routines to have the client bound to the appropriate server;

2. Passing control to the stubs when issuing and receiving an RPC for communication;

3. Having the stubs use the RPC Runtime routines to communicate with each other;

4. Basing the data transmission on the RPC protocols.

A DCE RPC between the application client and server is processed by various transport services provided by DCE and the network. Using Ethernet as an example, the entities that take part in the processing of RPCs for network transport are illustrated in Figure 3.1. The DCE application passes the RPC calls to the stubs, and the stubs issue RPC calls. The RPC calls, including the marshalled data, are decomposed by the RPC Runtime software into a series of Protocol Data Units (PDUs) according to the RPC protocols. The PDUs are then passed to the network transport layer, where they are further decomposed into TCP or UDP packets. Then the TCP/UDP packets are converted into IP packets by the IP layer, and are sent to Ethernet. On the receiver side, the Ethernet frames are received, and are reassembled to IP packets, to TCP/UDP packets, and to RPC PDUs. The RPC PDUs are reassembled to RPC data, and the RPC data is unmarshalled by the receiver's stub and passed to the application program. Among all the entities that participate in the RPC processing for transport, the RPC protocol is of vital importance.
3.4 The DCE RPC Protocols

3.4.1 Protocol Overview

When the application client and server make RPC calls that communicate over connection-oriented or connectionless transport, they do not and need not have an interface with the transport protocols to handle the transport of data. DCE's RPC Facility handles this by building RPC protocols on top of the transport protocols.

The DCE RPC protocol includes two parts: the connection-oriented RPC protocol and the connectionless RPC protocol. In DCE application development, these are called "ncacn" and "ncadg" respectively, as discussed in Chapter 2. These protocols use RPC Protocol Data Units (PDUs) for transmission. When the RPC PDUs for ncacn and ncadg RPC protocols are passed to the transport layer of the network, they are supported by the connection-oriented or connectionless transport protocols.
respectively (e.g. TCP or UDP over IP). At present the underlying transport protocols are limited to TCP/IP and UDP/IP, but the RPC protocol can be built on top of other transport protocols if needed.

An RPC PDU has at least two parts: the PDU header and the PDU body (as shown in Figure 3.2). If the PDU is for an authenticated RPC call, it must also contain an authentication verifier. This discussion deals with non-authenticated RPC PDUs.

![PDU Body]

**Figure 3.2: The Structure of an RPC PDU**

The PDU header contains the protocol control information. The body contains data. The header structures of the connection-oriented and connectionless RPC PDUs are different. Also, different types of connection-oriented PDUs have different header lengths. There are, in total, twenty types of RPC PDUs for connection-oriented and connectionless RPC protocols, and each has a type value related to it. For example, a “request” PDU for both connection-oriented and connectionless protocols has a type value of 0, while a “response” PDU has a type value of 1. A fragment acknowledgment ("fack") PDU for a connectionless protocol bears a type value of 9, while a “bind” PDU for a connection-oriented protocol bears a type value of 11.

The connectionless RPC protocol is built upon the UDP transport protocol. The connectionless RPC PDU has an 80 byte header, in which the control information is held. In this research, the control information, such as the PDU type, the sequence number and the fragment number, is the key to analyzing the network packets generated by the RPCs and RPC PDUs. For example, each of the PDUs generated from the same multi-PDU RPC call has the same sequence number, and a monotonically
increasing fragment number starting from zero.

The connection-oriented RPC protocol is built upon the TCP transport protocol. The operations required by the RPC Runtime, such as "establishing the connection", are performed by the transport protocol (TCP/IP). The connection-oriented PDUs have a structure similar to the connectionless ones, following the general PDU structure shown in Figure 3.2. There are header fields containing control information in every connection-oriented PDU. Some of the header fields are common for every type of connection-oriented PDU, and some of the header fields are specific to one type of PDU. Important information, such as the first and last PDUs of a multi-PDU transmission, are marked in the respective PDU headers. This helps the reassembly of data by the RPC protocol, and the analysis of network packets in this research as well.

The maximum size of an RPC PDU is defined by the implementations of the RPC protocols. There is also a definition for the minimum size in the standard. Usually, the client and server begin their communication by using the minimum-sized PDUs. A larger PDU size can subsequently be negotiated by the client and the server during their conversation. The maximum and minimum (if bigger than the standard defined one) sizes of a PDU can be different in different RPC implementations, and can be affected by the interoperation between different platforms that have different RPC versions installed. If the application client and server use the same RPC implementation, the negotiated PDU size is the maximum PDU size for this RPC implementation. If the application client and server run on different RPC implementations, the negotiated PDU size is the smaller maximum PDU size of the two DCE RPC implementations.
3.4.2 Fragmentation of the DCE RPC Data

The processing of an RPC can be explained as follows. The RPC data is marshalled by the client/server stub to obtain immunity from byte-ordering and other operating system problems. The stub then passes the data to the RPC Runtime. The RPC Runtime uses RPC protocols to fit the data into RPC PDUs. The PDUs that contain RPC data are always "request" and "response" PDUs. If the size of the RPC data is larger than the maximum PDU size and thus can not fit into one single "request" or "response" PDU, the data must be decomposed by the RPC protocols to form multiple PDUs. The PDUs are then passed to the network transport layer according to the type of RPC protocol (connectionless RPC protocol to UDP, and connection-oriented RPC protocol to TCP, respectively) where they are further decomposed into TCP/UDP packets. The TCP and UDP packets are then encapsulated in IP packets, which are finally sent to the Ethernet layer for transmission. The decomposition of data by different transport protocols is called data fragmentation. Two steps are involved in the data fragmentation: first, the data is fragmented into smaller packets according to the corresponding protocol; second, protocol headers are added to these data packets. Depending on the packet size defined by the protocol, sometimes the fragmentation process requires only the adding of headers. The data fragmentation caused by the different protocol layers shown in Figure 3.1 is called RPC fragmentation, UDP/TCP fragmentation, and IP fragmentation, respectively.

Whether or not fragmentation happens depends on the amount of data to be sent, and the sizes of the transmission units. For example, whether or not UDP/TCP fragmentation happens depends on whether the maximum size of the RPC PDU is larger than the size of a single UDP or TCP packet. In fact, the current DCE RPC PDU size is smaller than that of a UDP packet, so the UDP fragmentation happens only to add UDP headers to the RPC PDU data, then the data is passed to the IP layer. The TCP packet size is usually designed to fit into the data field of one
single IP packet; therefore TCP fragmentation happens to fragment the RPC PDU into TCP packets, and to add TCP headers to them. Then the IP fragmentation happens. The IP protocol fragments the UDP packets into multiple IP packets, and adds IP headers to them to form Ethernet frames. The IP protocol only adds IP headers to the TCP packets, which also form Ethernet frames. In some RPC implementations, the lower layer fragmentation is avoided by defining the maximum PDU size to be the size of data that can fit into a single IP packet; this is done because some think that IP fragmentation and reassembly are not reliable. Some argue that a larger PDU size will improve the performance of RPC (eg. throughput), and thus they use a large PDU size in their RPC implementation. In Digital DCE Ver. 1.3, the maximum RPC PDU size is 1464 bytes, while in IBM DCE for OS/2 Ver. 1.0, the maximum RPC PDU size is 4312 bytes (both for the connectionless RPC protocol).

Currently, most local area networks, including the DISCUS Lab where this research was conducted, are Ethernet-based. Using the RPC connectionless protocol on Ethernet as an example, the data units of communication from lower layers to higher layers are: Ethernet frame, IP packet, UDP data unit and RPC PDU, as shown in Figure 3.3. For the RPC implementations that define PDU size as the data size that can fit into a single IP packet, the multiple IP packets and Ethernet frames in the figure become single IP packets and single Ethernet frames.

When studying the PDU size and lower layer fragmentation, the size of the IP packet is important. In Figure 3.3, two IP packet sizes are shown. For this reason, this section will briefly discuss IP packet size.

The standard data size of an Ethernet frame (excluding the Ethernet header) is 1500 bytes [22] [26]. Every IP packet must fit into one Ethernet frame. In each IP packet, there are 20 bytes of IP header. Where interconnection of multiple subnetworks exists, there are 8 bytes of Logical Link Control (LLC) header in an IP
packet, but where there are no interconnections of sub-networks, the LLC header may not appear [21]. Therefore, the data an IP packet contains can be at most 1472 bytes (when sub-networks exist) or 1480 bytes (when sub-network do not exist).

In summary, the RPC Runtime processes the RPC data by fragmenting it into some larger number of smaller transmission units, as the data goes from higher layers to lower layers on the network, and by reassembling the small transmission units to larger ones as the data goes from lower network layers to higher ones. This procedure adds to the data amount to be transferred (headers) and to the processing time (to fragment and reassemble) on both the application client side and the application server side.

3.4.3 Flow Control

In any network-based environment, congestion in the network is unavoidable and can have unfortunate consequences. DCE also faces this problem. In the connection-oriented RPC protocol, flow control is managed by the TCP protocol on the transport layer of the network. Therefore, DCE is freed from considering flow control when it uses the connection-oriented RPC.
Because the connectionless RPC protocol is built upon the UDP transport protocol, and UDP does nothing to handle the congestion of the network, DCE has to take care of this problem. The connectionless RPC protocol applies a flow control algorithm to make the transfer of data more efficient and reliable. This algorithm is derived from Jacobson's "slow start" algorithm for congestion avoidance and control in the Internet [20]. The introduction of flow control to the connectionless RPC protocol, especially when transferring large amounts of data, affects the pattern of network traffic generated by multi-PDU RPC calls, and the performance of this kind of RPC.

Slow start is a scheme that lets the sender determine how much more traffic it can put on the network by checking the acknowledgments for the packets (PDUs) it sent previously. The rate the acknowledgments come back is affected by the congestion situation on the network. The sender begins by sending one PDU to the receiver. Upon receiving the acknowledgment of this PDU from the data receiver, the sender adds one to the size of its congestion window which controls the number of PDUs that can be sent out before getting the acknowledgment for the former PDUs from the receiver. Usually, when there is no congestion on the network, the congestion window size increases exponentially, until it reaches the prescribed upper bound for the congestion window size. If there is congestion on the network, and the sender fails to get the acknowledgment for a sent PDU, the congestion window stops growing. Figure 3.4 from [30] shows the idea of slow start.

Retransmission happens when a "packet loss" is detected. Because each PDU has a serial number, and PDUs are sent out serially, the packet loss is easy to detect. For the usual slow start algorithm, all the packets following the lost one will be transmitted again. For connectionless RPC, because both the client and the server side record and control the order of the fragments, only the lost PDU is retransmitted. When retransmission happens, the congestion window size shrinks to one.
Flow control also requires that on the server side, there is a buffer to hold the received PDUs, so that when the congestion window on the client side is big enough to allow the client to send a number of PDUs to the server in a short time period, the server still has somewhere to hold these PDUs without losing them. This buffer is called the receiving window. It is maintained by the server-side RPC Runtime. The PDUs held in the receiving window are later read out by the server process for further processing. There is also a limit (number of PDU) for the receiving window. In the acknowledgment for the received PDU, the server tells the client how much space is left in the receiving window. When the client gets the information about the receiving window, it counts those PDUs that have been sent out but not acknowledged. When the client detects that the receiving window is full, it stops sending PDUs to the server until it calculates from a new acknowledgment and determines that there are spaces in the receiving window.

Figure 3.4: Transmitting PDUs Under Slow Start (from [30])
3.5 Performance Aspects

RPC forms the basis of communication for DCE services and DCE applications, and thus how DCE and DCE applications perform depends heavily on how RPC performs. For example, if it takes a long time to make RPC calls, it takes a long time for the DCE client to get services from the server, and thus long waiting times for end users.

There is no doubt that the RPC Runtime processing influences the performance of RPC, and thus of everything that relies on RPC. How it affects RPC performance and what effects it has are the problems that are of interest in this research.

The RPC Runtime processing influences RPC performance first by data fragmentation. The RPC fragmentation on call data to form PDUs and the lower layer fragmentation on RPC PDUs to form UDP/TCP and IP packets both affect RPC performance by adding to the amount of data that must be transferred (by adding headers at each level), and by introducing delay at the time of fragmentation and reassembly. Second, the PDU-level flow control in the connectionless RPC protocol affects the performance of RPC because it decides when a PDU is to be sent out.

By studying the RPC protocols, data fragmentation and flow control, the following factors are considered to affect RPC performance:

- RPC PDU size. This factor is determined by the RPC implementation, and varies every time the client and server start to communicate with each other. The PDU size decides how the data fragmentation happens at each transmission layer. By affecting the data fragmentation, it also affects the characteristics of the workload that RPC puts on the network.

- Maximum congestion window size (for connectionless RPC only). The congestion window size determines how many PDUs can be sent to the network...
back-to-back. If this size is too small, some PDUs have to wait for the acknowledgment before they are sent out, even if there is no congestion on the network.

- Retransmission policy (also for connectionless RPC's flow control). For example, if the loss of one PDU causes the retransmission of all the following PDUs, the time taken for one RPC call may be increased substantially.

- Maximum size of the receiving window on the server side. When the RPC data amount (or, data waiting to be sent out) exceeds the value of N, where:

\[ N = \text{Max. receiving window size} \times \text{Max. PDU size} \]

and there is not a lot of congestion on the network so that the client has a large (even maximum) congestion window, it is possible that the client is prevented from sending all the PDUs comprising one DCE RPC call and must wait for space in the receiving window. This causes the round trip time (RTT) of RPC to increase.

Since these factors can not be changed from outside RPC's implementation, they are considered as internal factors in this investigation. In studying the internal factors, it must be noticed that this effect is sometimes influenced by the external factors. Taking the receiving window size as an example, the maximum size of the server receiving window and the maximum client congestion window size in a single RPC implementation are chosen carefully. Therefore, the situation when there are more PDUs in the congestion window than the receiving window can hold seldom exists. But when interoperation happens, things get more complicated. If the size of a PDU decreases, and the number of PDUs comprising a single RPC call increases, while the server receiving window size and client congestion window size do not change, then the situation described above can exist. If the receiving window problem exists, it may also hide the effects of some other external factors, for example, the network load. That is, whether or not there is congestion on the
network, the client has to wait for space in the receiving window. The time taken is much longer than the network contention, and the RTTs under both network load are not much different.

3.6 Summary

This chapter described how client/server communications in DCE are supported by RPC at run time. A brief description of the RPC protocols was followed by a discussion of the structure and size of the RPC Protocol Data Unit. Data fragmentation by the RPC and lower layer network protocols, and flow control in connectionless RPC, are two processes that affect RPC performance.

The data transmitted between the DCE application client and server is marshalled by the stub codes and passed to the RPC Runtime. The Runtime processing includes the RPC fragmentation of the data by the RPC protocols, and the lower layer fragmentation by the UDP/TCP and IP protocols. How and when fragmentation happens depends on the RPC PDU size, which is defined by the RPC implementation and determined by negotiation between the client and server hosts at the beginning of the communication. The fragmentation of data adds to both the amount of delay and the amount of data to be transferred, and therefore plays an important role in RPC performance.

The next chapter reviews other performance studies of DCE and DCE RPC.
Chapter 4

Related Studies of DCE and DCE RPC Performance

4.1 Introduction

As a trade-off for easy distribution, DCE introduces an entire layer of software facilities between the applications and the supporting systems/networks. The performance of the DCE components directly affects the performance of DCE applications running on top of DCE. As DCE becomes more widely available, performance studies are becoming more common. This chapter reviews a number of recent studies of the performance of DCE components.

4.2 Studies of DCE Performance

DCE is made up of five basic components that provide five kinds of distributed services. The performance of each of these distributed services has been studied.

Measurement studies of the Cell Directory Service (CDS) component are presented in [28], [11] and [42]. The CDS is both application-related and management-related, and is used in two situations: first, when the application client binds to the server; and second, when the administrators or the users run the CDS control programs.
Martinka et al. [28] presented a systematic study of OSF DCE 1.0.1 CDS performance in an environment with HP/Apollo 9000/720 workstations running HP-UX 8.07 as the operating system. They noticed that two types of CDS access are basic in running DCE applications – the RPC binding imports of the client application from the CDS (retrieving the CDS information), and the RPC binding exports from the application server to the CDS (the modification of Name Space). They studied the factors that affected CDS access response time and found that the cdsclerk cache had a major impact on CDS access response time.

Corn et al. [11] measured CDS performance for IBM DCE for OS/2 Ver. 1.1 (based on OSF DCE 1.0.2) in an environment containing IBM PS/2 machines running the OS/2 operating system, connected by a 16 Mbps token-ring local area network. They studied the response time of selected commands from the CDS Control Program. Response times of both reading and writing commands (create, delete, list, show) upon both directories and objects were measured. They found a highly linear relationship between the response times of CDS control program commands and the number of CDS objects or directories. The effect of client-side caching on the response times of CDS access commands was also reported in this study.

The performance of OSF DCE 1.1 CDS for different CDS configurations was studied by Russell [42], for the purpose of handling ten thousand users' needs in a large DCE cell. Different hardware configurations of the CDS server running AIX or OS/2 Warp as operating systems with a variety of OS/2, AIX and Windows DCE clients were tested. The server configurations include one primary CDS server with one, three, six, and eight secondary CDS servers (IBM RS/6000, 486-66, 486-33, and 486-33, respectively). The primary CDS servers were 486-66, Pentium 90, RS/6000 models 570, 580 and 990. The study concluded that CDS performs better when there are more secondary servers in the configuration. The limits of CDS capacity for different configurations were also measured.
The performance of the DCE Security Service (DSS) has been studied from two perspectives: authenticated RPC (or security RPC), and security administration. The former is always discussed in the context of RPC performance, and thus is left to the next section.

Corn et al. [11] studied the DSS as a core DCE component, and studied its performance by counting the number of user authentication, registration, login and access control commands. This study is similar to their study of CDS in that the response times of a group of key DSS commands were measured. The experimental environment consisted of IBM PS/2 machines running OS/2 operating system, as DCE security clients and servers.

Russell [42] studied the dce_login performance from the security server's perspective for a large cell with over ten thousand users. Different security server hosts were measured to decide which is capable of providing the required DSS service in such a cell. The performance of three different OS/2-based DCE Security servers was measured: 486-33, 486-66, and Pentium 90. The cell contained various kinds of DCE clients. The DSS performance was expressed in terms of "DCE logins per second". This study found a 2 percent reduction on maximum DSS performance for each additional 1000 accounts, when the number of accounts exceeded 1000. A comparison of the effect of using CDS and the file system to store the security server's address was also reported. Russell found that looking up the Security server's address put heavy loads on the CDS server, and affected the performance of Security commands. If the DCE clients were allowed to store the Security server's address in a local file, the response time of DCE login could be reduced. This study also found that DCE login's user response time depended upon the processor speed of the DCE client.

DCE cell performance for different network topology in a large heterogeneous environment was reported by Russell [41]. This study compared the performance of DCE services and DCE applications in a large cell and those in two smaller cells.
Application performance when using either a single DCE server or multiple DCE servers to support DCE services and applications in a cell was studied by Perry et al. [34]. The servers in this study were IBM RS/6000 and IBM PS/2's, running AIX and OS/2 as the operating systems, respectively. This study found that moving the workload to multiple servers is an effective way of controlling the application server bottleneck, and recommends a multiple server configuration when planning a cell.

Other studies of DCE performance include modelling and monitoring of both DCE applications and DCE services. Techniques for modelling the performance of applications running on top of DCE were reported in [37] and [25], and applications of these models to predict the performance of DCE applications were reported in [36]. Modelling of RPC was reported in [39] and [24]. Other studies have been based on data obtained from instrumenting DCE and DCE applications. Instrumentation can be done either at the stub level, by modifying the IDL compiler [9], or at the RPC Runtime level, by developing a group of measurement routines in the RPC Runtime library [16]. Integrating the modelling and monitoring infrastructure to predict and manage the performance of both DCE and DCE applications was reported in [15], [24] and [17].

4.3 Studies of DCE RPC Performance

Because of the important role RPC plays in DCE, RPC performance has been the focus of several recent studies. In these studies RPC performance was measured in different environments, and the effects of different system factors were studied. RPC response time (expressed as round trip time, or RTT) was the basis of some investigations ([11], [13], [23], [24]), and RPC throughput was the basis of others ([41], [11] and [35]). Some of these studies also considered the effect of multi-threading on RPC performance ([23], [24], [13]).
Round trip time (RTT) is the most commonly used measure of (end-to-end) RPC performance. RTT is defined as the time elapsed between the client's issuing the RPC call and the client's receiving the response from the server. The RTT is related to the data amount contained in the RPC call in the former studies. *RPC throughput*, defined as "bytes of data transferred by RPCs per second", describes the capacity of RPC in transferring data. Because different RPC calls contain different amounts of data, the "baseline" throughput is defined by the number of null RPC calls per second.

Measurements reported in the literature are typically based on synthetic RPC applications or benchmark programs. The most commonly used application is simple data transfer. The client issues an RPC call containing from zero to X bytes of data (where X varies from 1K bytes to the amount that the researchers consider to be adequate for their studies, eg. 24K bytes, 40K bytes, or 64K bytes), and the server, upon receiving the request RPC call from the client, invokes the routine requested by the client, and returns with a response when the routine is finished. To focus on the RTT of the RPC call, the routine is usually designed to be null.

Measuring the round trip time of RPCs that contain different amounts of data was the starting point and focus of many RPC performance studies. The important issue that concerns this measurement is how RPC performance is affected by the amount of data it contains. Because the first RPC always does some binding-related work, the first RPC and subsequent RPCs were distinguished in the studies. The binding method chosen in the applications affects the performance of the first RPC, but has no impact on the subsequent RPCs.

Dasarathy *et al.* [13] measured RPC response time, defined as "time elapsed between the time just before the RPC is made and the time just after the RPC is
completed"\(^1\) in a homogeneous environment consisting of a set of HP RISC workstations running DCE 1.0.1. In this environment, both the client and the server were the same kind of workstation, with the same or different MIPS rate. They measured the RPCs that were generated by a synthetic data-transfer application, and distinguished the first RPC and subsequent RPCs in their measurement. They observed a linear relationship between the RPC response time and the data size it contains, with the increasing rate of 2 milliseconds per kilobyte of RPC data in their environment. Also, with an explicit binding method, the response time of the first RPC was four times as long as the average response time of the subsequent RPCs.

Khandker et al. [23] also measured RPC round trip times. In their study, the RPC calls measured were generated by a synthetic data transfer application with from zero to 64K bytes of data. The cell contained a set of IBM RS/6000 workstations running the operating system AIX 3.2.4 and OSF DCE 1.0.2. They were connected by a 10Mbps Ethernet. The chosen RPC protocol was connectionless RPC over UDP/IP. String binding was used to eliminate the communication of the first RPC to CDS and to the endpoint map on the server. The data was passed by RPC calls as fixed size arrays. In their environment, they observed a non-linear increase of the RPC round trip time with the amount of data contained in the RPC calls, for both first RPCs and subsequent RPCs. With the same amount of data, the RTT of the first RPC is longer than that of subsequent RPCs. Figure 4.1 shows the result from this measurement. On the RTT-vs.-data size curve, sharp increases of RTTs were found for data sizes of 1393, 4017, 8033, 12049, 16065, 20081, 24097, 32129, 48193 and 64257 bytes. This was attributed to the decomposition of data to IP packets – every sharp increase of the RTT was caused by the additional network packets generated by the data.

Corn et al. [11] reported measurements on the RTTs of RPCs with different

\(^1\)This definition is similar to that of the RPC round trip time – "the time elapsed between the client's issuance of the RPC call and the client's receipt of the response from the server".
amounts of data in an environment containing IBM PS/2 machines running the OS/2 operating system and IBM DCE for OS/2 Version 1.0 (based on OSF DCE 1.0.2). They observed a linear increase in RTT as the RPC data size increases. They also observed that the response time increases sharply between 2K bytes and 4K bytes for connection-oriented RPC, and between 4K and 8K bytes for connectionless RPC. They deduced from this observation that the maximum size of the connection-oriented RPC PDU was between 2K bytes and 4K bytes, while that of the connectionless RPC PDU was between 4K bytes and 8K bytes. The factor of RPC protocol was also observed to affect RPC performance. They observed that connection-oriented RPC performs better for small calls (< 2K bytes), while connectionless RPC performs better for large calls. In fact, the larger the data transferred via a connectionless RPC call, the better the performance. They explained this by the “slow start” policy of flow control introduced by the connectionless RPC protocol on top of the UDP transport. Figure 4.2 compares the RTT-vs.-data size curves from the connectionless and connection-oriented RPC protocols.

The effect of system heterogeneity on RPC performance was reported by Dasarathy et al. [13]. They noticed that when client and server machines had different MIPS
rates, the RPC round trip time increased. While this kind of heterogeneity in hardware platforms is very common in the environments of RPC performance studies, heterogeneity in operating system platforms has not been reported in any previous RPC performance studies.

Much attention has been paid to the performance of RPC when it is working with other DCE services—in particular, the Threads service and the Security service. A performance impact of threading was reported by Dasarathy et al. [13]. They found that one multi-threaded application client gave better RPC performance than the same number of single-threaded application clients. Threading was also studied by Khandker et al. [23], who measured RPC throughput over different numbers of client threads, and the round trip times of RPCs sent over different numbers of client threads were measured by Khandker and Teorey [24]. The impact of the Security service has also been studied. The Security service provides protection for data contained in authenticated RPCs. Different levels of protection exist. Measurements of the round trip times of Authenticated RPCs over different protection levels were reported in [11] and [28], both of which found that higher security levels led to slower RPCs.
Finally, DCE RPC performance has also been studied in large scale, multi-user systems. Perry et al. [35] measured RPC throughput in a large multi-user environment containing a number of IBM PS/2's (running the OS/2 operating system) and an IBM RS/6000 (running the AIX operating system). The hosts were connected by a 16 Mbps token-ring local area network, and the application workload was generated by three kinds of benchmark programs. This study found that when the number of clients increased, connectionless RPC yielded higher throughput than connection-oriented RPC, but when the number of clients was small (≤ 10), the two kinds of RPC yielded almost identical throughput. They also found that the RPC throughput for higher authentication levels was lower despite the number of clients.

4.4 Factors Affecting DCE RPC Performance

RPC performance is affected by a number of factors, including the characteristics of the hardware/software environment, the nature of the DCE application, and the traffic on the network. The factors that are determined by the implementation of RPC are considered as “internal factors”. For example, the PDU size and the size of the congestion window in flow control are internal factors. “External factors”, such as the network congestion, the chosen transport, and the amount of data contained in an RPC call, affect RPC performance by influencing the levels of internal factors, also directly. Some of the interesting external factors that affect RPC performance are described as follows.

Because RPC relies on the network to communicate, the type, quality and load of the network play an important role in performance. Data fragmentation happens when the RPC data is transferred on the network. The data contained in a single RPC call is decomposed ultimately into Ethernet frames. When the load on the Ethernet is high, the spacing between these frames can be increased, and some of these frames may get lost because of congestion on the network. This directly increases the response time for RPC communications, and may decrease the throughput.
The network factor also includes the transport layer protocols with which the RPC protocols work. TCP and UDP are the two transport protocols upon which the current RPC implementations are built. Other connectionless or connection-oriented transport protocols will probably be considered by later RPC implementations. Because of the different nature of these transport protocols, the performance of RPC built upon them varies. The two available transport protocols, TCP and UDP, affect the internal factors in several ways: the different PDU sizes, the layer that flow control happens on (for RPC over TCP, it happens at the TCP layer, on IP packets, for RPC over UDP, it happens at the RPC layer, on RPC PDUs), and the congestion window size. For example, for large amounts of data, the number of IP packets that wait to be transferred can always be larger than the size of congestion window in a transmission over TCP, while the number of PDUs generated by the same amount of data is smaller than the congestion window size for RPC over UDP. Delays are introduced for the IP packets that are waiting to enter the congestion window. So if the size of data contained in an RPC call exceeds a certain value, RPC over UDP may perform better than RPC over TCP.

Heterogeneity in the environment is also an external factor. Heterogeneity affects RPC performance because of factors such as the different byte ordering of different platforms, which affects the marshaling and unmarshalling time, and because of the negotiation and change of the RPC PDU sizes when two different DCE implementations communicate with each other.

The nature of the data transferred by RPC also affects RPC performance. In typical application environments, the amount of data transferred by RPCs is small (< 2K bytes), which causes a smaller load on the network. In other applications, such as those that transfer multimedia data (video frames, etc.), or fetch large numbers of rows from a database, there are RPCs with large data volumes that cause heavier traffic on the network.
4.5 Summary

This chapter reviewed some related performance studies on DCE and DCE RPC. Using RPC round trip time as the performance metric, the impact of factors such as the amount of data contained in an RPC call, security levels, multithreading, and the transport protocols on which the RPC protocols are built has been studied. Other performance issues that have been investigated, but are not reviewed in this chapter, include the nature of multithreading, caching in the first RPC, and disconnected or partially connected client/server communication.

There remain many factors that influence RPC performance that have not yet been thoroughly studied. These include the effects of external factors such as network factors and system heterogeneity. Also, the way external factors affect RPC performance via internal factors such as the data fragmentation and flow control needs more investigation.

The next chapter will introduce the experimental work that forms the basis of this thesis. The results are presented in chapters 6 and 7.
Chapter 5

The Experimental Work

5.1 Introduction

The goal of this research, as was mentioned already, is to gain improved understanding of DCE RPC by studying how it is structured and how it works. The purpose of the experimental work is to explore the factors that affect RPC performance, in order to understand how RPC performs, and to measure the packet arrival process of DCE RPC, to study how to model it, and to provide some insights into the characteristics of the workload on the network caused by DCE RPC. Three factors are addressed: the protocol sequence that the RPC communication uses, the load on the network, and interoperation across operating systems.

This chapter first describes the purpose of the experiments. The experimental setup and method are described next, along with the performance metrics involved in this research. These are followed by a description of the experimental design for the modelling work.

5.2 The Purpose of the Experimental Work

The main aim of the work is to study DCE RPC performance and DCE RPC-caused network traffic experimentally. The experimental work consists of measurement and modelling.
The measurement work first concerns three system/network factors that affect DCE RPC performance. As mentioned in Chapter 4, a number of factors can influence the performance of RPC in the DCE environment. Among these factors, three are recognized as important external factors: the RPC protocols used, the network load, and the interoperation of different operating systems. Understanding these factors will help in recognizing those that most strongly impact DCE RPC performance. Measuring DCE RPC performance at different levels of these three factors helps in understanding how DCE RPC performs, which is an important aspect of understanding DCE RPC.

The other part of the experimental work is to measure the packet arrival process of the DCE RPC-caused network traffic, and make an effort to model it. This part of the experiment concerns the internal factors that affect RPC performance, as discussed in Chapter 4. The two internal factors are the fragmentation caused by the DCE RPC and underlying protocols, and the flow control strategies. The measurement helps in characterizing the workload generated by DCE RPC on the network. Based on the workload characterization, a workload model can be built, to explain the measurements, and to provide more insights into DCE RPC. Whether DCE RPC workload can be adequately described by a simple model will be investigated and discussed.

The contributions of the study are to be:

1. To understand the RPC protocols and the relevant internal details, such as fragmentation and flow control, that affect DCE RPC performance and the characteristics of the DCE RPC traffic on the network.

2. To measure how DCE RPC performance is affected by factors such as the volume of network traffic, interoperation across operating systems, and different RPC protocols.
3. To characterise the RPC-caused network traffic by studying the arrival process of the transmission units, and to investigate candidate packet arrival models with the purpose of building a workload model for DCE-RPC-based applications of the future.

5.3 Measurement Methodology

This section describes the experimental methods. The experimental environment is described first, followed by the trace collection method employed.

The experimental environment is made up of three parts: the DCE cell, the DCE applications running on the cell, and the network monitor and measurement tools. Figure 5.1 shows the entire experimental environment. Traces are collected in this environment using the measurement tool, and are stored for off-line analysis.

![Figure 5.1: The Experimental Environment](image-url)
5.3.1 The DCE Cell

The experimental work was done on a single DCE cell with six computers connected by a 10 Mbps Ethernet local area network in the Distributed Systems Performance Laboratory at the University of Saskatchewan. Configuration details are given in Figure 5.1. Only four of the machines in the cell were involved in the experiment. The machines marked as “A” and “B” in the figure are IBM ThinkPADs, which are part of the cell but were not involved in this research. The four machines that took part in the experiment were as follows:

- The cell server: A DEC Alpha 2100A-500MP with three 190MHz ALPHA AXP 21064 CPUs and 256MB of main memory. It runs the Digital UNIX V3.2C operating system, and DCE V1.3 for OSF/1. The host name is Skorpio3.

- A cell client: A DEC Alpha 2100A-500MP with a single 190MHz ALPHA AXP 21064 CPU and 128MB of main memory. It runs the Digital UNIX V3.2C operating system and DCE V1.3 for OSF/1. The host name is Alf.

- A cell client: An IBM Personal System 350 with a Pentium 75 processor and 16MB RAM. It runs IBM’s OS/2 WARP Connect operating system (version 3), and IBM DCE for OS/2 version 2.1 Beta. The host name is Morty.

- A cell client: An IBM PS/2 Model 80 with a Reply 486dx2-66 motherboard and 16MB of RAM. It runs IBM’s OS/2 WARP Connect operating system (version 3), and IBM DCE for OS/2 version 2.1 Beta. The host name is Ferdie.

All the machines can run as either an application client or an application server. For simplicity, we refer to the machines by their host names. When describing the application client/server pairs, the operating system a machine runs is also used to name the computer platform. Thus, Skorpio3 and Alf are called Digital UNIX machines, and Morty and Ferdie are called OS/2 machines.
In this DCE cell, because there are two different types of computers (IBM Personal Computers and Digital Alphas), two implementations of DCE are involved, each with different RPC implementations. These implementations of RPC use different PDU sizes, and when they interoperate with each other, they negotiate on the PDU size. Observations on the system have determined the PDU sizes in this environment. They are shown in Table 5.1 for the different client-server combinations.

Table 5.1: Upper Bounds of PDU Sizes (in bytes) in Different RPC Implementations

<table>
<thead>
<tr>
<th></th>
<th>TCP OS/2</th>
<th>TCP Digital UNIX</th>
<th>UDP OS/2</th>
<th>UDP Digital UNIX</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS/2</td>
<td>4352</td>
<td>-</td>
<td>4312</td>
<td>-</td>
</tr>
<tr>
<td>Digital UNIX</td>
<td>4096</td>
<td>4096</td>
<td>1464</td>
<td>1464</td>
</tr>
</tbody>
</table>

5.3.2 The DCE Application

A synthetic DCE application was used in the experiments. There are three reasons for using synthetic application programs:

1. To keep the factors that affect the application behaviour under control. These factors include the data size contained in an RPC call, the binding method used by the application, and the protocol sequences the server listens on.

2. To focus the study on the performance of DCE RPC, instead of the performance of the applications.

3. To use them instead of "real" DCE applications, because real DCE applications are currently rare.

The application was a standard DCE application with a UUID interface definition, client/server code, and remote procedure code. The copies of the same application on different platforms bear the same UUID for the purpose of studying
the effects of interoperability.

The application can do the data transfer in different ways, as the user requests. The method by which data is transferred is chosen by the user when running the application client. The application server and remote procedure are not affected by the user choice on the client side.

The application server listens on both protocol sequences: the connectionless RPC protocol sequence `ncadg_ip_udp`, and the connection-oriented RPC protocol sequence `ncacn_ip_tcp`. The application server prepares the string_binding information and prints it out when starting execution. The remote procedure merely performs a procedure return when it receives the RPC call. A header routine `errchk.h` checks the errors whenever the application calls an RPC Runtime routine.

Two remote procedures are defined in the interface definition file: `send_null()` and `send()`. `send_null()` issues a null RPC from the client to the server; `send()` issues an RPC call with 1K bytes to 24K bytes of request data from the client to the server in a fixed-size array.

If the user chooses to transfer the data in a "random" way, the application client randomly chooses the data size contained in the RPCs it issues, from a set of data sizes: 0, 1000, 1500, 2000, 3000, 4000, 6000, 8000, 10000, 14000, 18000, and 24000 bytes. This is what the synthetic applications did in previous performance studies of DCE RPC. The user is also able to choose the percentage of RPCs containing no data, and data small, medium and large in size in all the RPCs that are sent out. He/she can give the data sizes and the percentages to the program at the time it begins to run. For example, the user can choose to send 10% of the RPCs as null RPCs, 70% of the RPCs containing 1000 bytes of data, 15% of the RPCs containing 2000 bytes of data, and the remaining 5% of the RPCs containing 40000 bytes of data. With this choice, the synthetic application is able to emulate a "real"
application more closely, because most of the "real" RPCs contain only data that is small in size. If the user chooses to transfer the data in a "fixed" way, he/she can decide the size of the data that is to be transferred, within the range of 0 to 65535 bytes. Then all the RPCs sent out contain the same amount of data.

The application can run multi-threadedly. The maximum number of concurrent threads is eight. When it is needed to increase the workload on the system, multiple copies of the application can run concurrently, and with different user choices.

5.3.3 The Workload

The workload to be measured (and modelled) is generated by the synthetic DCE application described in the previous section. A single run is described as follows:

- The client submits 10,000 RPC calls to the server and receives replies from the server for all the RPC calls it submits.

- Each of the 10,000 RPC calls has a data size (in bytes) chosen randomly from the set 0, 1000, 1500, 2000, 3000, 4000, 6000, 8000, 10000, 14000, 18000, 24000. Each of these sizes is generated with equal probability.

5.3.4 Tools

The measurement station shown in Figure 5.1 is a Sun SPARC Station IPC, running tcpdump version 2.1. It is not in the DCE cell, but is connected in the same Ethernet segment. Tcpdump is a passive network monitoring tool that captures the packet headers (and data part, if needed) from the Ethernet. The captured headers (or data) are filtered to display those of interest to the users. A time stamp is recorded for each captured packet. The advantage of this approach to measurement is that the measurement tool resides on a machine other than the machines sending and receiving the network packets, and thus does not interfere with the communications being measured. But if the machine that is running tcpdump is slow compared to
the generation of monitored traffic, some packets may be lost, which can make the measurement less accurate.

In addition to the measurement station running tcpdump, there is a hardware network monitor which is also located on the same Ethernet segment as the DCE application clients and servers. This is a HP 4972A LAN Protocol Analyser. It is used for two purposes: to monitor the traffic on the network, and to generate background traffic on the network for the purpose of getting different network load levels. When using the LAN Protocol Analyser to generate network traffic, the network utilisation, the generated frame size, and the size of the frame burst can be controlled. Figure 5.2 shows a display of the monitored and controlled parameters of the LAN Protocol Analyser.

Figure 5.2: Table Display of the Traffic Generator
5.3.5 Levels of Factors in the Measurement Environment

Three factors affecting DCE RPC performance were identified for the measurement experiment. They are: RPC protocols, network load, and interoperation between different platforms, i.e. client/server pairs. The levels of these factors are defined as follows.

The two RPC protocols in DCE are connectionless RPC (RPC over UDP) and connection-oriented RPC (RPC over TCP). Whether the measurement is on connectionless or connection-oriented RPC is determined by nature of the DCE application.

There are two kinds of host running different operating systems and different versions of DCE on top of different hardware platforms. They are: OS/2 machines and Digital UNIX machines. There are four combinations of client/server pair if the two kinds of machines are used as DCE application client and server. These are shown in Table 5.2 (and are named as “client-to-server”). When a Digital UNIX server and an OS/2 client were employed, errors occurred because the server was running a lower version of DCE. Because these runs could not complete, this client/server pair is not taken into consideration in this research.

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS/2</td>
<td>Digital UNIX</td>
</tr>
<tr>
<td>Digital UNIX</td>
<td>Digital UNIX-to-OS/2</td>
</tr>
<tr>
<td></td>
<td>(Errors)</td>
</tr>
<tr>
<td>OS/2-to-OS/2</td>
<td>Digital UNIX-to-Digital UNIX</td>
</tr>
</tbody>
</table>

Another factor affecting DCE RPC performance is network load. First of all, a relatively “quiet” network is required to measure the DCE RPC performance when there is little impact from other network traffic. Then some other load needs to be put on the network in order to determine the effects of more network load from other
sources on the performance of DCE RPC. The experimental environment is part of a live network with some of the DCE nodes playing other roles for other users. In fact, the two Digital UNIX machines are also file servers supporting students, staff and faculty members of the Computer Science Department, which means that the load levels of the experimental network are not always controllable. The relatively “quiet” network was assumed to be found during the off-peak period of network usage, and the network with other load can be generated by either “real” users, or by inducing “synthetic” load with the LAN Protocol Analyser. The advantage of using load generated by real users is that the load is realistic, but the level of the load can not be controlled. On the other hand, using the LAN Protocol Analyser to generate traffic when the network is quiet, can provide a range of load levels from 20% to 100%, while the packet lengths and bursts are only simple simulations. In this experiment, LAN Protocol Analyser-generated load was used for the OS/2-OS/2 pair, and user-generated load was used for the Digital UNIX-to-OS/2 pair and the Digital UNIX-to-Digital UNIX pair, because there is such a small number of users on the network between the OS/2 machines that a regular network load can not be found.

The levels of network load are presented as follows:

- Relatively quiet network:
  - Light network load: the load on the network during the “off-peak” period of network usage. The load level reading from the LAN Protocol Analyser is 10% ± 3%. The same load level is used for all three client/server pairs

- Loaded network:
  - Regular network load: the load on the network when there are a normal number of users working on the network. This level is used for Digital UNIX-to-OS/2 and Digital UNIX-to-Digital UNIX pairs
  - Heavy network load: the load on the network that is generated by the LAN Protocol Analyser during an off-peak period of network usage. This
load level reading from the LAN Protocol Analyser is 60% ± 3% before the DCE application starts running. This level is used only for OS/2-to-OS/2 pair

5.3.6 Trace Collection

The traces are grouped by the experiments to which they correspond. The traces in one group result from the different levels of a single factor that is under study, and replications of these traces. The DCE application runs on the application client and server machines, and tcpdump runs on the measurement station to capture the traffic caused by the execution of the application.

Because the traces are collected in a relatively controlled environment, and the synthetic application runs in a controlled manner, the size of the trace needs not be large to give all the information needed, with acceptable accuracy. The traces in this experimental work are usually 10 to 15 Mbytes in size, containing the network packets generated by 10000 RPC calls. Off-peak periods of network usage are considered to be:

- Midnight to early morning of every weekday;
- Friday nights;
- Daytime of Saturdays and Sundays

For the Digital UNIX-to-OS/2 pair and the Digital UNIX-to-Digital UNIX pair, traces when “regular load” is placed on the network were collected during:

- Any time between 10:00am and 11:30am of weekdays
- Any time between 2:00pm and 4:00pm of Monday to Thursday

Because those traces under regular network load for the Digital UNIX-to-OS/2 pair and the Digital UNIX-to-Digital UNIX pair are collected when the network load is provided by daily users on the network, they are heavily influenced by the
behaviour of these users. Because user behaviour changes, the conditions under which different traces are collected also changed. For example, if when trace1 is collected all users are doing small read/write operations, but when trace2 is collected, somebody is sending a huge file to the internet, trace1 and trace2 may provide data under different network load; however, both are treated as traces under "regular load". This user behaviour is not controllable because it happens in a live environment. Moreover, although other traces were collected from the off-peak periods of network usage, disturbance from other users can not be avoided totally. System maintenance happens in off-peak periods, and the workload caused by these operations could affect the measurement. Also, according to the information given by tcpdump, the packet loss rate of tcpdump is observed to be always around 0.05% which is considered to be within acceptable limits for this work. Because the packets captured by tcpdump are IP packets, on which nothing marks whether they belong to a certain RPC PDU except the PDU header that appears in their data fields, the analysis program is unaware of the packet loss unless the lost packet is the first IP packet in a PDU (which bears the PDU header). All of the above are the sources of error to the results, and occur randomly. Replications of traces were used to reduce the impact of these random errors on the results, by looking for consistency across traces, and by calculating the average of mean values when necessary.

5.4 The Performance Metrics

To study the arrival process of different transmission units, the interarrival times of these units are measured. The end-to-end latency of RPC is described by the RPC round-trip time (RTT), and the capacity of the RPC service is described by throughput. The interarrival times of different transmission units used in the experiments are defined as follows, together with the RPC RTT and throughput.

- Interarrival time:
- **Inter-IP time** is the elapsed time between the arrival of one IP packet and the arrival of the next IP packet\(^1\). This metric describes the spacing between consecutive IP packets appearing on the network. The transmission unit in this level is IP packet, and is referred to henceforth as "IP".

- **Inter-PDU time** is the time elapsed from the arrival of the first IP packet in one RPC PDU to the arrival of the first IP packet in the next RPC PDU generated by the same application. This metric describes the spacing between consecutive RPC PDUs appearing on the network, that are generated by the same application. The transmission unit in this level is RPC PDU, and is referred to henceforth as "PDU".

- **Intra-PDU-IP time** is the time elapsed between consecutive IP packets that resulted from a single RPC PDU. The transmission unit in this level is called "IP-in-PDU", to distinguish it from IP packets generated by different PDUs. Similarly, IP-in-Call's are those IP packets comprising one DCE RPC call. The inter-call-IP time was not measured in this research.

- Speed and capacity of RPC service:

  - **Round-trip time (RTT)** is defined as the time elapsed between the client issuing the RPC call and receiving the entire response from the server.

  - **RPC throughput** is the data-transfer throughput of RPC, which is defined as bytes of data transferred by RPCs per second\(^2\).

Figure 5.3 shows the definitions of interarrival times and RTT.

---

\(^1\)All measurements of interarrival times are with respect to a fixed observation point on the network, the measurement station.

\(^2\)The baseline throughput of DCE RPC is normally defined as number of null RPCs per second, but the baseline throughput is not used in this research.
5.5 Modelling the DCE RPC-caused Network Workload

Workload study is a significant aspect of any performance study of computer systems, and DCE is no exception. Without understanding the workload caused by DCE RPC, it is hard to understand, analyze, and predict the performance of DCE RPC accurately, and thus, by extension, the performance of DCE. The measurement of DCE RPC reveals some of the characteristics of RPC-caused workload and provides some understanding of the DCE RPC protocols, and knowledge of the characteristics of RPC-caused workload helps in building DCE RPC workload models.

At the same time, the work related to modelling, such as choosing the parameters, helps researchers understand RPC activity more deeply.

For the purpose of understanding DCE RPC, although the measurement is based
on synthetic DCE applications, it is still useful to do some modelling work based on the measurement results, especially on the packet arrival process. Because the amount of data contained in an RPC call is chosen synthetically, statistics on the call length, such as the number of IP packets and RPC PDUs generated by a single RPC call, make little sense in the workload study. The attempt to model the workload focuses on modelling the arrival process of the transmission units, as described in the previous section.

It is likely that the process that determines the spacing between the network packets generated by the RPCs will not be same for all units. Because of RPC fragmentation and the fragmentation by the lower layer network protocols, and the flow control policy applied on different layers of network protocols, packets are sent out with different spacings between them. Whether or not a simple model, such as a single-parameter Poisson model, is capable of modelling the network packet arrival process is investigated in this research. In [26], a two-level model called the *packet tandem model* was used to describe the Ethernet workload generated by NFS. The fragmentation of NFS data by the NFS protocol and the underlying network protocols is similar to the fragmentation of RPC data by DCE RPC protocols and the lower layer network protocols. Therefore, it is possible that a model similar to the packet tandem model can describe the RPC-caused network workload.

The modelling work investigates the application of one-level and multi-level models in the modelling of the packet arrival process of IP packets on Ethernet caused by running DCE RPC applications. The number of levels considered for this multi-level model was 2. There are in fact more than two levels in preparing the DCE RPC data before it is sent out: the first is the call level, where the user data is fit into an RPC call; the second is the PDU level, where the call data is fragmented into RPC PDUs; the third is the IP level, where the RPC PDUs are fragmented into IP packets. There might be even more levels, such as the user application (which determines when the data communication is taken place). There are reasons for not
taking these levels into account in this research:

- The user behaviour, such as how and when they run the DCE applications, is beyond the scope of this research

- The experimental application is a synthetic one, so the amount of data in an RPC call in this application doesn't represent any real application behaviour. Also for this reason, the call length or even PDU length can not be modelled

- A two-level model can be a representative of multi-level models, because the primary objective is to determine whether or not a single level model is sufficient in the workload modelling. If a two-level model works better, it proves that a one-level model is NOT sufficient

- The PDU level and IP level measurements are relatively unaffected by the application used, and so represent some of the workload characteristics well; therefore the PDU and IP levels provide good upper and lower levels in a two-level model

The models generate the inter-IP times by sampling from two different distributions: the exponential distribution and the hyper-exponential distribution. These two distributions are chosen for the following reasons:

1. The single-level model that derives from sampling the inter-arrival time from an exponential distribution is a Poisson model. The Poisson model is a well-known and commonly-used simple workload model, the application of which is worth investigating. Comparing the two-level exponential model with the one-level exponential model shows the effect of changing the structure of the model from one-level to two-level.

2. The hyper-exponential distribution is chosen because it takes the variance of inter-IP times as an input parameter. It is a combination of a few exponential distributions, and so represents more bursty traffic. This is useful in modelling the DCE RPC traffic on the network when the network load is heavy so that
the measured variance of interarrival times of the transmission units is large enough that a hyper-exponential distribution describes the characteristics of the traffic better than does an exponential distribution.

3. Comparing a one-level hyper-exponential model with two-level models is more convincing than is using only an exponential distribution in the one-level model.

4. Using a hyper-exponential distribution in one or both level(s) in a two-level model might give better modelling results if the traffic is bursty on this level.

With two structures for the model (one-level and two-level), and two distributions (hyper-exponential and exponential), a total of five combinations are investigated: one-level exponential (O.L.E.), one-level hyper-exponential (O.L.H.), two-level exponential exponential (T.L.E.E.), two-level hyper-exponential exponential (T.L.H.E.), and two-level hyper-exponential hyper-exponential (T.L.H.H.). The first distribution (for example, the H in T.L.H.E. model, i.e. the upper level of the two-level model) is the distribution used to generate the inter-PDU time. The second distribution (the E in T.L.H.E. model, i.e. the lower level of the two-level model) is the distribution that generates the interarrival time of IP packets comprising one PDU (intra-PDU-IP times). Measured values, such as the mean and variance of inter-PDU times, intra-PDU-IP times, and inter-IP times, are supplied as parameters of the workload models. The measured mean, variance, and distribution of inter-IP times are used to verify the modelled values, and to evaluate the models. Two functions from the C-SIM simulation package, exp() and hyper(), are called to sample the interarrival times from exponential and hyper-exponential distributions. Details on the parameters of the models and the method of sampling from the two-level model to get modelled inter-IP times are described in Chapter 7.
5.6 Summary

This chapter described the purpose, methodology and performance metrics used in the experimental work. The purpose of the experimental work in this research is to study the arrival process of different transmission units in RPC communication and to measure the performance of RPC in order to understand the characteristics of the RPC-caused network traffic, and the factors that influence RPC performance. The goal was to investigate what kind of model is suitable for describing the RPC-caused network traffic.

The experimental environment is part of the Computer Science Department network, containing a DCE cell with heterogeneous hardware platforms, operating systems and DCE versions, a monitoring node and a measurement station, all connected by a 10 Mbps Ethernet. The DCE applications used in the experiments are synthetic programs using the RPC facility.

Groups of similar traces were collected. To minimize the disturbance of the network traffic caused by other users, the traces were collected in off-peak periods of network usage. Also, to minimize the possible influence to the experimental results by other traffic on the network, and the packet loss when collecting the data, replications of traces were used in the analysis.

The performance metrics involved in the experimental work are the inter-IP time, inter-PDU time, and the intra-PDU-IP time, for the study of packet arrival process, and the RPC round-trip time and the RPC throughput for the measurement of RPC latency and capacity.

Work relating to the modelling of the packet arrival process of DCE RPC-caused network traffic consists of investigating one-level and two-level models with exponential and hyperexponential distributions, using measured interarrival characteristics.
as parameters. Measured and modelled inter-IP time statistics and distributions are compared. This work will suggest which kind of model is more suitable in describing the DCE RPC-cause network traffic.

The next chapter presents the experimental results, and the analysis of these results.
Chapter 6

The Experimental Results

6.1 Introduction

This chapter presents the experimental results and analyses of the results. There are two parts in the measurement work: the measurement of characteristic features of the packet arrival process of RPC-caused network traffic, and the measurement of RPC latency and capacity, both in the DCE environment. The results for these two parts of the experiments are presented and analysed separately.

The first section of this chapter presents the results from the measurement of the packet arrival process of DCE RPC-caused traffic on the network. Measuring the packet arrival process is part of the workload characterisation of DCE RPC on Ethernet. Statistics on the interarrival times of transmission units – IP packets, PDU-IP’s, and DCE PDUs are analysed: mean values to show the average interarrival times, variances to show the scope of change of the interarrival times, and coefficients of variation (CoVs, calculated as standard deviation divided by mean) to show the degree of burstiness of the traffic. Full distributions of the interarrival times are also presented to give a more detailed picture of the interarrival times. It is found that the DCE RPC-caused network traffic is bursty, i.e. the IP packets on Ethernet appear as PDU-related bursts, and arrivals of IP packets are highly dependent on one another. The dependency and burstiness of the traffic are caused by the actions of the RPC protocols.
The second section presents the results from the measurement of DCE RPC latency and capacity. Mean RPC round trip times and throughputs are presented in plots and bar charts. It is found that the round trip time of DCE RPC increases with the amount of data contained in the RPC, almost linearly, with jumps where the first PDU appears. Increasing network load causes RPC Round trip times to increase, but the influence of network load is much smaller than that of the RPC data size.

There are some results that are at odds with expectations. These results are worth studying because they may suggest improvements in the design of RPC protocols. The analysis of unexpected measurement results is presented in the third section.

6.2 Measurements of the Packet Arrival Process

The purpose of measuring the packet arrival process of DCE RPC-caused network traffic is to understand the characteristics of the RPC workload on the network which connects the nodes of a DCE environment. There are significant differences between the RPC protocols (i.e. connection-oriented RPC, also called RPC over TCP, and connectionless RPC, also called RPC over UDP). These differences are caused by different PDU sizes, and the differences in their flow control mechanisms. There are two major groups of measurements in studying the packet arrival process of RPC-caused network traffic according to the RPC protocols, and they are presented separately in the following sections.

6.2.1 The Packet Arrival Process of Connection-oriented RPC

When using the connection-oriented RPC protocol to communicate, different client/server pairs (OS/2-to-OS/2, Digital UNIX-to-OS/2, and Digital UNIX-to-Digital UNIX)
use different PDU sizes. For the above three kinds of client/server pairs, the PDU sizes are 4352, 4096 and 4096 bytes, respectively. However, there are two facts that all these three client/server pairs share. First, for the data sent by the application, both RPC and IP fragmentation happen. Second, flow control happens at the IP level, is managed by the TCP protocol, and has little impact on the packet arrival process of RPC PDUs. These important facts about connection-oriented RPC suggest that the packet arrival processes for all these client/server pairs bear similar characteristics.

Network load also affects the pattern of traffic caused by DCE RPC. To isolate the influence of other network traffic, the packet arrival process of different RPC transmission units is first measured under light network load. This measurement is called a "basic measurement" in this study. After some insight on the RPC packet arrival process is gained from the basic measurement, the impact of network load is analysed, followed by discussion of the impact of interoperation between different systems.

The Basic Measurement

For each client/server pair, four traces were collected under light network load. The results from all 12 traces are summarized in Tables 6.1, 6.2, and 6.3.

Table 6.1 shows the mean interarrival times of different transmission units of all three client/server pairs. Table 6.2 shows the variance of the interarrival times, and Table 6.3 shows the CoVs. The following are observed from all the trace data (comparisons are made within a trace and no cross-client/server-pair comparisons are made):

1. Among all three interarrival times, the intra-PDU-IP time is the smallest. It is much smaller than the inter-IP time. The inter-PDU time is the largest. This observation means that the spacing between the IP packets that are generated from the same PDU is smaller than the spacing between the IP packets from
Table 6.1: Mean Interarrival Times (msec) Under Light Network Load for Connection-oriented RPC

<table>
<thead>
<tr>
<th>Client/Server Pair</th>
<th>Transmission Unit</th>
<th>Trace 1</th>
<th>Trace 2</th>
<th>Trace 3</th>
<th>Trace 4</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS/2-to-OS/2</td>
<td>PDU</td>
<td>8.870</td>
<td>8.840</td>
<td>8.788</td>
<td>8.976</td>
<td>8.868</td>
</tr>
<tr>
<td></td>
<td>IP-in-PDU</td>
<td>1.341</td>
<td>1.322</td>
<td>1.304</td>
<td>1.354</td>
<td>1.330</td>
</tr>
<tr>
<td></td>
<td>IP</td>
<td>1.722</td>
<td>1.677</td>
<td>1.716</td>
<td>2.365</td>
<td>1.870</td>
</tr>
<tr>
<td></td>
<td>IP-in-PDU</td>
<td>1.093</td>
<td>1.094</td>
<td>1.134</td>
<td>1.123</td>
<td>1.111</td>
</tr>
<tr>
<td>Digital UNIX-to-OS/2</td>
<td>PDU</td>
<td>6.001</td>
<td>7.379</td>
<td>7.177</td>
<td>7.460</td>
<td>7.004</td>
</tr>
<tr>
<td></td>
<td>IP</td>
<td>2.269</td>
<td>2.790</td>
<td>2.714</td>
<td>2.821</td>
<td>2.649</td>
</tr>
<tr>
<td></td>
<td>IP-in-PDU</td>
<td>0.997</td>
<td>1.023</td>
<td>1.015</td>
<td>1.020</td>
<td>1.014</td>
</tr>
</tbody>
</table>

Table 6.2: Variances of Inter-arrival Times Under Light Network Load for Connection-oriented RPC

<table>
<thead>
<tr>
<th>Client/Server Pair</th>
<th>Transmission Unit</th>
<th>Variances of Inter-arrival Times</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Trace 1</td>
</tr>
<tr>
<td>OS/2-to-OS/2</td>
<td>PDU</td>
<td>41.508</td>
</tr>
<tr>
<td></td>
<td>IP</td>
<td>29.103</td>
</tr>
<tr>
<td></td>
<td>IP-in-PDU</td>
<td>1.771</td>
</tr>
<tr>
<td>Digital UNIX-to-Digital UNIX</td>
<td>PDU</td>
<td>53.478</td>
</tr>
<tr>
<td></td>
<td>IP</td>
<td>21.248</td>
</tr>
<tr>
<td></td>
<td>IP-in-PDU</td>
<td>0.836</td>
</tr>
<tr>
<td>Digital UNIX-to-OS/2</td>
<td>PDU</td>
<td>36.880</td>
</tr>
<tr>
<td></td>
<td>IP</td>
<td>16.941</td>
</tr>
<tr>
<td></td>
<td>IP-in-PDU</td>
<td>0.204</td>
</tr>
</tbody>
</table>

different PDUs. This is consistent with expectations.

2. The variance of intra-PDU-IP times is the lowest, while the variance of inter-PDU times is the highest. This observation shows that the spaces between the IP packets comprising one PDU are quite uniform.

3. The CoV is high for inter-IP times, but low for inter-PDU times and intra-PDU-IP times. This observation shows that the IP traffic on the Ethernet is bursty, but the bursts are not found within PDUs and among PDUs. In other
Table 6.3: CoVs of Interarrival Times Under Light Network Load for Connection-oriented RPC

<table>
<thead>
<tr>
<th>Client/Server Pair</th>
<th>Transmission Unit</th>
<th>CoVs of Inter-arrival Times</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Trace 1</td>
</tr>
<tr>
<td>OS/2-to-OS/2</td>
<td>PDU</td>
<td>0.726</td>
</tr>
<tr>
<td></td>
<td>IP</td>
<td>1.542</td>
</tr>
<tr>
<td></td>
<td>IP-in-PDU</td>
<td>0.993</td>
</tr>
<tr>
<td>Digital UNIX-to-Digital UNIX</td>
<td>PDU</td>
<td>1.606</td>
</tr>
<tr>
<td></td>
<td>IP</td>
<td>2.677</td>
</tr>
<tr>
<td></td>
<td>IP-in-PDU</td>
<td>0.837</td>
</tr>
<tr>
<td>Digital UNIX-to-OS/2</td>
<td>PDU</td>
<td>1.012</td>
</tr>
<tr>
<td></td>
<td>IP</td>
<td>1.814</td>
</tr>
<tr>
<td></td>
<td>IP-in-PDU</td>
<td>0.454</td>
</tr>
</tbody>
</table>

In other words, the IP bursts on the Ethernet are related to the arrival of PDUs.

The observation of small and uniform spacing between IP packets comprising one PDU can be explained by the fact that there is no delay at the IP-in-PDU level except for putting the IP packet to the network. Observations on the inter-arrival time statistics show that the RPC PDUs have the largest and most variant interarrival times. The RPC PDUs generated by the application used in this experiment can contain from one to three IP packets, and the multi-IP PDUs have much larger interarrival times than the single-IP PDUs. Also, fragmentation of PDUs to IP packets takes time. Observations also show that the PDU traffic is not bursty. Because the network load in this experiment is very low, there is no contention on the network, and no delay is introduced by flow control; meanwhile, the application is designed to generate RPC calls back-to-back, so the RPC PDUs are put to the network back-to-back without additional delays other than preparing the PDUs.

Figures (a), (c), and (e) in Figure 6.1, Figure 6.2 and Figure 6.3 give the distribution of interarrival times of the transmission units of connection-oriented RPC under light network load for the three client/server pairs. As expected, the intra-PDU-IP times tend to be concentrated in a small region on the plot, showing the
small spacing between IP-in-PDUs and non-burstiness of the traffic. (This kind of observation is called “small-region concentration” of interarrival time henceforth.) Comparing to the distribution plot of intra-PDU-IP time, there is a longer tail on the inter-IP time distribution plot, which results from the larger variance of inter-IP times, and an obvious second peak on the inter-PDU time distribution plot, which is caused mainly by the “inter call delay”, the time for preparing a new RPC call at the application level. The observations are consistent with the observations made from the tables previously, and are consistent across all three client/server pairs.

The Impact of Network Load

Increasing network load will cause contention when sending network packets, and increase the interarrival times of all transmission units. What impact it has on the traffic pattern is the focus of this section. There are two perspectives in studying the impact of network load on the packet arrival process of DCE RPC. First, the network load was synthetically generated to be very heavy (more than 90% when the DCE RPC application load is added, and 60% if no DCE application running). This made the impact of network load on the interarrival times measurement quiet clear. This experiment was done with the OS/2-to-OS/2 pair. Second, in real systems, the network load could not be very heavy all the time, but with more bursts compared to synthetic loads. The impact of this kind of network load (regular load) is also of interest, and was investigated with the Digital UNIX-to-Digital UNIX and Digital UNIX-to-OS/2 pairs.

Table 6.4 shows statistics of interarrival times of different transmission units under heavy network load for OS/2-to-OS/2 pair. Figure 6.4 compares the measurement results under light and heavy network loads for the OS/2-to-OS/2 pair. In order to present both statistics in the same chart, standard deviations are used instead of variances. More detailed comparisons of distributions of interarrival times are shown in Figure 6.1. Figures (a), (c), and (e) are the interarrival time distributions under light network load, while Figures (b), (d), and (f) are the interarrival
Figure 6.1: Interarrival Time Distributions of the OS/2-to-OS/2 Pair Under Different Network Loads for Connection-oriented RPC
Table 6.4: Interarrival Time Statistics for OS/2-to-OS/2 Pair Connection-oriented RPC Under Heavy Network Load

<table>
<thead>
<tr>
<th>Transmission Unit</th>
<th>Statistics</th>
<th>Trace 1</th>
<th>Trace 2</th>
<th>Trace 3</th>
<th>Trace 4</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDU</td>
<td>Mean</td>
<td>14.973</td>
<td>15.657</td>
<td>12.480</td>
<td>13.955</td>
<td>14.266</td>
</tr>
<tr>
<td></td>
<td>Var</td>
<td>375.952</td>
<td>427.498</td>
<td>205.955</td>
<td>246.167</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>CoV</td>
<td>1.295</td>
<td>1.320</td>
<td>1.150</td>
<td>1.124</td>
<td>-</td>
</tr>
<tr>
<td>IP</td>
<td>Mean</td>
<td>4.852</td>
<td>5.161</td>
<td>4.607</td>
<td>4.857</td>
<td>4.869</td>
</tr>
<tr>
<td></td>
<td>Var</td>
<td>116.232</td>
<td>131.381</td>
<td>85.185</td>
<td>91.226</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>CoV</td>
<td>2.222</td>
<td>2.221</td>
<td>2.003</td>
<td>1.966</td>
<td>-</td>
</tr>
<tr>
<td>IP-in-PDU</td>
<td>Mean</td>
<td>2.551</td>
<td>2.735</td>
<td>2.281</td>
<td>2.474</td>
<td>2.510</td>
</tr>
<tr>
<td></td>
<td>Var</td>
<td>52.536</td>
<td>51.184</td>
<td>28.367</td>
<td>35.044</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>CoV</td>
<td>2.842</td>
<td>2.616</td>
<td>2.335</td>
<td>2.393</td>
<td>-</td>
</tr>
</tbody>
</table>

time distributions under heavy network load.

From the table and figures, the following observations are made:

1. The mean interarrival times increase for all transmission units when the network load increases. The mean intra-PDU-IP time increases but is still smaller than the mean inter-IP time, which means that the spacing between consecutive IP packets comprising the same PDU is still smaller than that between the last and first IP packets from consecutive PDUs. Comparison of the distribution of interarrival times shows that more than 30% of IP packets are delayed because of the increased network load.

2. With the increased network load, the interarrival times of all transmission units become more variant. Longer tails are found on the plots of interarrival time distributions, and the standard deviations increase.

3. The CoV of inter-IP times becomes very large, indicating the presence of bursty IP traffic on the network. The CoVs of the inter-PDU and intra-PDU-IP times are also increased. The CoV of intra-PDU-IP time time is larger than that of inter-IP time shows the congestion control works on IP packets.
4. Interesting observations are made from the plots of interarrival time distributions. The small region concentration of inter-PDU time no longer exists, while the small region concentration of intra-PDU-IP time still remains, even under such a heavy network load.

Increased network load causes increased contention at the IP level. Meanwhile, because the flow control of the connection-oriented RPC is done at the IP level, delays are also introduced in inter-IP times by the congestion on the network. The IP level delay causes increasing inter-PDU time and intra-PDU-IP time. Both network contention and flow control make the interarrival times more variant. In spite of the contention and congestion on the network caused by the increased network load, IP packets comprising one PDU tend to come more closely spaced because it takes more time to prepare a new PDU than it does to simply put the IP packets onto the network.

Table 6.5: Interarrival Time Statistics for Digital UNIX-to-Digital UNIX Pair Connection-oriented RPC Under Regular Network Load

<table>
<thead>
<tr>
<th>Transmission Unit</th>
<th>Statistics</th>
<th>Trace 1</th>
<th>Trace 2</th>
<th>Trace 3</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDU</td>
<td>Mean</td>
<td>6.589</td>
<td>6.831</td>
<td>6.021</td>
<td>6.480</td>
</tr>
<tr>
<td></td>
<td>Var</td>
<td>393.307</td>
<td>281.853</td>
<td>63.104</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CoV</td>
<td>3.010</td>
<td>2.458</td>
<td>1.319</td>
<td></td>
</tr>
<tr>
<td>IP</td>
<td>Mean</td>
<td>2.491</td>
<td>2.583</td>
<td>2.278</td>
<td>2.451</td>
</tr>
<tr>
<td></td>
<td>Var</td>
<td>149.482</td>
<td>98.700</td>
<td>26.393</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CoV</td>
<td>4.907</td>
<td>3.847</td>
<td>2.255</td>
<td></td>
</tr>
<tr>
<td>IP-in-PDU</td>
<td>Mean</td>
<td>1.500</td>
<td>1.558</td>
<td>1.250</td>
<td>1.436</td>
</tr>
<tr>
<td></td>
<td>Var</td>
<td>9.953</td>
<td>26.207</td>
<td>2.845</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CoV</td>
<td>2.104</td>
<td>3.287</td>
<td>1.349</td>
<td></td>
</tr>
</tbody>
</table>

Tables 6.5 and 6.6, and Figures 6.2 and 6.3, show the measurement under regular network load and the comparisons of interarrival time distributions under light and regular network load. Although the regular network load is only slightly heavier than the light network load, differences in the interarrival time statistics and distributions can be observed. The observation under regular network load is consistent with that
Table 6.6: Interarrival Time Statistics for Digital UNIX-to-OS/2 Pair Connection-oriented RPC Under Regular Network Load

<table>
<thead>
<tr>
<th>Transmission Unit</th>
<th>Statistics</th>
<th>Trace 1</th>
<th>Trace 2</th>
<th>Trace 3</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDU</td>
<td>Mean</td>
<td>7.271</td>
<td>8.031</td>
<td>8.281</td>
<td>7.861</td>
</tr>
<tr>
<td></td>
<td>Var</td>
<td>73.771</td>
<td>67.453</td>
<td>265.030</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>CoV</td>
<td>1.181</td>
<td>1.023</td>
<td>1.966</td>
<td>-</td>
</tr>
<tr>
<td>IP</td>
<td>Mean</td>
<td>2.749</td>
<td>3.037</td>
<td>3.131</td>
<td>2.972</td>
</tr>
<tr>
<td></td>
<td>Var</td>
<td>32.293</td>
<td>29.415</td>
<td>105.498</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>CoV</td>
<td>2.067</td>
<td>1.786</td>
<td>3.280</td>
<td>-</td>
</tr>
<tr>
<td>IP-in-PDU</td>
<td>Mean</td>
<td>1.237</td>
<td>1.569</td>
<td>1.510</td>
<td>1.439</td>
</tr>
<tr>
<td></td>
<td>Var</td>
<td>1.910</td>
<td>14.318</td>
<td>4.351</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>CoV</td>
<td>1.118</td>
<td>1.789</td>
<td>1.381</td>
<td>-</td>
</tr>
</tbody>
</table>

made under heavy network load. With increased network load, the means, variances, and CoVs of interarrival times are increased, the small region concentration on the interarrival time distribution remains for the intra-PDU-IP time, and no longer exists for the inter-PDU time.

The Impact of Interoperation

When interoperation happens, although the DCE application client and server are on the same network segment, the DCE RPC performance numbers are different for different client/server pairs. The causes of the differences include: different machine speeds, different operating systems, and different DCE versions. The effects of operating system and DCE version are most evident during the binding period, and once the negotiation is finished, it no longer affects the communication. Under connection-oriented RPC, all three client/server pairs investigated communicate using PDU sizes that are bigger than one IP packet, and both PDU and IP fragmentation happened during the period of communication. Figure 6.5 shows the mean interarrival times and their standard deviations from one trace of each client/server pair. Once again, standard deviations are used so that they can be easily presented on the same chart as the mean values.
From the plot, it can be observed that the mean inter-PDU time of the Digital UNIX-to-Digital UNIX pair is the smallest, and the inter-PDU time of OS/2-to-OS/2 pair is the largest, and the same is true with the inter-IP times. This is easy to explain because the Digital UNIX machines are much faster than the OS/2 machines. Thus it takes less time to prepare the data before putting it onto the network on the Digital UNIX machines.

The interesting observation is that the mean interarrival times of PDU and IP for the Digital UNIX-to-OS/2 pair are not the largest, although interoperation happens between these machines. In fact, they fall in between the mean interarrival times of the two non-interoperating pairs. The performance of the Digital UNIX-to-OS/2 pair can be explained in one of two ways: either changing the client of the OS/2-to-OS/2 pair to a faster machine improves RPC performance, or changing the server of the Digital UNIX-to-Digital UNIX pair to a slower machine decreases the RPC performance. Because the OS/2-to-Digital UNIX pair DCE doesn't work (lower version of DCE on the server side), it can't be determined whether the processing time of the client machine or the server machine affects the RPC performance more profoundly.

The most important observations on the characteristics of the network traffic caused by connection-oriented RPC are summarized as follows:

1. The IP packets on the network come as PDU-related bursts, with interarrival times within the burst much more constant than is the case for all the inter-IP times. This observation hold for different client/server pairs.

2. Increased network load causes contention when sending the network packets, and increases the interarrival times of the transmission units. Nevertheless, the PDU-related bursts of IP packets are still present despite other network load.
3. Interoperation between different platforms/operating systems affects RPC performance. Whether the speed and capacity of the client machine or the server machine affect RPC performance more profoundly is not known.

6.2.2 The Packet Arrival Process of Connectionless RPC

It is expected that the RPC protocols used for communication would affect the characteristics of DCE RPC workload on the network. For connection-oriented RPC, flow control happens at the TCP layer and affects the rate by which individual IP packets are put to the network. Therefore the effect of flow control can hardly be seen at the PDU level. For connectionless RPC, on the other hand, flow control is done at the PDU level. This means that an entire PDU is presented for transmission by the RPC layer, and then all the IP packets comprising it are put to the network as quickly as possible. The effect of flow control at the PDU level would result in higher variance in the inter-PDU times.

In the client/server pairs studied in this research, the OS/2-to-OS/2 pair uses a PDU size of 4312 bytes for connectionless RPC. The Digital UNIX-to-Digital UNIX pair uses 1464-byte-PDUs. When interoperation happens between the two systems, the smaller PDU is used for communication. Each PDU is then fitted into one single IP packet and put to the network. For large PDUs (in the OS/2-to-OS/2 pair), two levels of fragmentation happen. For small PDUs (in the other two client/server pairs), only RPC fragmentation happens, and there are other performance concerns such as flow control.

The Basic Measurement and Unexpected Observations

Similar to the study on connection-oriented RPC, a basic measurement is done first when the network load is light.
Figure 6.6 shows the statistics of interarrival times of different transmission units for OS/2-to-OS/2 pair. Tables 6.7, 6.8, and 6.9 summarize the measurement of interarrival times of connectionless RPC under light network load.

The following observations are made:

1. For the OS/2-to-OS/2 pair, as expected, the mean interarrival times decrease as the data move closer to the Ethernet. The intra-PDU-IP time is much smaller than the inter-IP time. The intra-PDU-IP time is 0 for the Digital UNIX-to-Digital UNIX pair because there is only one IP packet in each PDU. And for the same reason, the inter-IP time and inter-PDU time are identical for the Digital UNIX-to-Digital UNIX pair.

2. For the OS/2-to-OS/2 pair, the inter-PDU time is the most variant, and the intra-PDU-IP time is the least variant. The inter-PDU time is even more variant than the inter-IP time, and this is not expected comparing to the results of connection-oriented RPC.

3. The CoV values show that both PDU and IP-in-PDU traffic are not bursty for the OS/2-to-OS/2 pair. In both cases the CoV values are smaller than 1.0, with the CoV value for the intra-PDU-IP time being slightly smaller. For the Digital UNIX-to-Digital UNIX pair, the PDU and IP packet sizes are identical, and the interarrival time for PDUs is as variant and bursty as that for IP packets.

For the OS/2-to-OS/2 pair the observation that the inter-PDU time is non-bursty but very variant is quite strange. This is inconsistent with expectations and with measurements from connection-oriented RPC. More details can be found from the distributions in Figure 6.7.

On the inter-PDU time distribution plot, there is a big second peak which almost doubles the first one in width. This means that a large number of PDUs (near 60%) have been “delayed” (excluding the long but narrow tail that appears on the
Table 6.7: Mean Interarrival Times (msec) Under Light Network Load for Connectionless RPC

<table>
<thead>
<tr>
<th>Client-Server Pair</th>
<th>Transmission Unit</th>
<th>Trace 1</th>
<th>Trace 2</th>
<th>Trace 3</th>
<th>Trace 4</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IP</td>
<td>3.906</td>
<td>3.982</td>
<td>3.914</td>
<td>3.942</td>
<td>3.936</td>
</tr>
<tr>
<td></td>
<td>IP-in-PDU</td>
<td>1.157</td>
<td>1.211</td>
<td>1.150</td>
<td>1.152</td>
<td>1.167</td>
</tr>
<tr>
<td>Digital UNIX-to-Digital Pair</td>
<td>PDU</td>
<td>2.099</td>
<td>2.207</td>
<td>2.296</td>
<td>2.098</td>
<td>2.175</td>
</tr>
<tr>
<td></td>
<td>IP</td>
<td>2.099</td>
<td>2.207</td>
<td>2.296</td>
<td>2.098</td>
<td>2.175</td>
</tr>
<tr>
<td></td>
<td>IP-in-PDU</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>IP-in-PDU</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Table 6.8: Variance of Interarrival Times Under Light Network Load for Connectionless RPC

<table>
<thead>
<tr>
<th>Client-Server Pair</th>
<th>Transmission Unit</th>
<th>Trace 1</th>
<th>Trace 2</th>
<th>Trace 3</th>
<th>Trace 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS/2-to-OS/2 Pair</td>
<td>PDU</td>
<td>31.498</td>
<td>38.376</td>
<td>34.767</td>
<td>35.850</td>
</tr>
<tr>
<td></td>
<td>IP-in-PDU</td>
<td>0.312</td>
<td>1.558</td>
<td>0.284</td>
<td>0.284</td>
</tr>
<tr>
<td>Digital UNIX-to-Digital Pair</td>
<td>PDU</td>
<td>43.315</td>
<td>46.665</td>
<td>61.604</td>
<td>21.464</td>
</tr>
<tr>
<td></td>
<td>IP</td>
<td>43.315</td>
<td>46.665</td>
<td>61.604</td>
<td>21.464</td>
</tr>
<tr>
<td></td>
<td>IP-in-PDU</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Digital UNIX-to-OS/2 Pair</td>
<td>PDU</td>
<td>4227.036</td>
<td>4354.343</td>
<td>5544.742</td>
<td>6081.148</td>
</tr>
<tr>
<td></td>
<td>IP</td>
<td>4227.036</td>
<td>4354.343</td>
<td>5544.742</td>
<td>6081.148</td>
</tr>
<tr>
<td></td>
<td>IP-in-PDU</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

plot). Considering the fact that this measurement was done under light network load when no congestion appears on the network, and so no PDUs have to wait for the congestion window to expand, this second peak looks bizarre. In fact, even if there were congestion on the network, the flow control affects only a very small percentage of PDUs, and the second peak should not have been so big.

It is assumed that UDP works well, because it is an established protocol (which
Table 6.9: CoV of Interarrival Times Under Light Network Load for Connectionless RPC

<table>
<thead>
<tr>
<th>Client-Server Pair</th>
<th>Transmission Unit</th>
<th>CoVs of Inter-arrival Times (Unit)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Trace 1</td>
</tr>
<tr>
<td>OS/2-to-OS/2 Pair</td>
<td>PDU</td>
<td>0.567</td>
</tr>
<tr>
<td></td>
<td>IP</td>
<td>1.325</td>
</tr>
<tr>
<td></td>
<td>IP-in-PDU</td>
<td>0.483</td>
</tr>
<tr>
<td>Digital UNIX-to-Digital UNIX Pair</td>
<td>PDU</td>
<td>3.135</td>
</tr>
<tr>
<td></td>
<td>IP</td>
<td>3.135</td>
</tr>
<tr>
<td>Digital UNIX-to-OS/2 Pair</td>
<td>IP-in-PDU</td>
<td>0.0</td>
</tr>
</tbody>
</table>

also supports communication with the Department's file system). Considering that a fundamental difference between connectionless and connection-oriented RPCs is the level at which flow control happens, it is reasonable to suggest that the second peak comes from the congestion window somehow failing to expand properly, making about 60% of the PDUs wait for acknowledgment of previous PDUs before being sent out. In the synthetic application used, the data sent out is (0, 1000, 1500, 2000, 4000, 6000, 8000, 10000, 12000, 14000, 18000, 24000) bytes in size. This generates 12 first PDUs and 18 subsequent PDUs in total. Because the data size is chosen randomly, about 60% of the PDUs are “subsequent” PDUs. If the maximum size of the congestion window were set to 1 PDU, which is very possible because the DCE version used is a beta release, then only the 40% “first” PDUs were sent out without delay (especially when under light network load), while the remaining 60% of the PDUs failed to get into the congestion window, and had to wait for the acknowledgment for the previous PDU to come back.

There is also a smaller but obvious second peak on the inter-IP time distribution plot. This peak is related to the second peak on the inter-PDU time distribution plot. On the distribution plot of intra-PDU-IP time, almost all the spacings between
the IP packets comprising one PDU are very small, and also relatively constant. This observation is consistent with that made for connection-oriented RPC, and suggests that although some PDUs are delayed, the RPC-caused network traffic is still bursty, with the IP packet bursts related to PDUs.

The Effects of Network Load and Interoperation

When the network load was increased, the experiments on the OS/2-to-OS/2 pair encountered problems. When the network utilisation was increased to 60% (before the DCE application was started), the connection-oriented RPC worked well between the OS/2 machines, while connectionless RPC aborted execution in the middle of the run every time. Even decreasing the network utilisation to 25% failed to stop the DCE application from aborting execution. This problem seems to result from the increasing network load because it did not happen when the network utilisation was around 10%, with peaks of utilisation under 20%. It appears that the application terminated because it could not receive some awaited acknowledgments for the sent-out PDUs during a specific period of "timeout".

Table 6.10: Statistics of Inter-arrival Times of PDU/IP With Regular Network Load for Connectionless RPC

<table>
<thead>
<tr>
<th>Client-Server Pair</th>
<th>Statistics</th>
<th>Variances of Inter-arrival Times</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Trace 1</td>
</tr>
<tr>
<td>Digital UNIX-to-Digital UNIX Pair</td>
<td>Mean</td>
<td>5.991</td>
</tr>
<tr>
<td></td>
<td>Var.</td>
<td>222.341</td>
</tr>
<tr>
<td></td>
<td>CoV</td>
<td>2.489</td>
</tr>
<tr>
<td>Digital UNIX-to-OS/2 Pair</td>
<td>Mean</td>
<td>14.482</td>
</tr>
<tr>
<td></td>
<td>Var.</td>
<td>18333.787</td>
</tr>
<tr>
<td></td>
<td>CoV</td>
<td>9.350</td>
</tr>
</tbody>
</table>

Figure 6.8 and Figure 6.9 show the interarrival time plots under different network load for the Digital UNIX-to-Digital UNIX pair and the Digital UNIX-to-OS/2 pair, respectively. Table 6.10 shows interarrival time statistics taken from these two
client/server pairs under regular network load. For these pairs, the PDU fits into a single IP packet, the measured interarrival times can be called either inter-PDU times or inter-IP times. The following observations are made:

1. Comparing the results under light and regular network load, the mean interarrival times, the variances and the coefficients of variation all increased with increasing network load.

2. For the lightly loaded network, the majority of IP packets came within short time intervals. These IP packets are probably generated by the same RPC call. The longer tails on the interarrival time distribution plots under regular network load indicate that network contention delayed some of the transmission units.

3. The differences between the measurement results from the “lightly loaded network” and the “regularly loaded network” were not large because there was daily maintenance workload during the off-peak hours, which is NFS traffic over UDP/IP, so that the network is not sufficiently quiet. Nevertheless, the differences are still noticeable.

When interoperation happens, the first thing that is done before the communication begins is the negotiation of PDU size – the smaller of the two communicating systems’ PDUs is taken. In this experiment, the Digital UNIX-to-OS/2 pair uses single-IP PDUs to communicate. Because each RPC PDU generates only one IP packet, flow control happens at the PDU level but still works on IP packets. That is, flow control works with the same unit for both RPC protocols. Therefore the inter-IP time distributions should look the same for the two protocols, as happened with the Digital UNIX-to-Digital UNIX pair.

From Tables 6.7, 6.8, 6.9, 6.10, and Figure 6.9, the following observations about interoperation under connectionless RPC can be made:
1. The tail of the interarrival time distribution plot of the Digital UNIX-to-OS/2 pair is longer than that of the Digital UNIX-to-Digital UNIX pair, because interoperation used more time to marshall and unmarshall the data sent and received at the beginning and end of a RPC call.

2. Under light network load, the Digital UNIX-to-OS/2 pair has the largest mean inter-IP time of all three client/server pairs. This observation is different from that made for connection-oriented RPC, and is unexpected. The Digital UNIX-to-OS/2 pair has a faster client than the OS/2-to-OS/2 pair, and so the mean inter-IP time of the Digital UNIX-to-OS/2 pair is expected to be smaller. This observation is odd, and it may relate to another odd observation made about the variances of inter-IP times as described in the next point.

3. An extremely large variance is observed from the statistics of inter-PDU and inter-IP times of the Digital UNIX-to-OS/2 pair. The variance is more than a thousand times that of the Digital UNIX-to-Digital UNIX pair, although it was expected to be in the same range.

   This observation indicates that something must have happened to introduce large delays for some PDUs. This matches an observation obtained in measuring the RPC RTTs, and is discussed in Section 6.3, along with the RTT abnormalities.

   The most important observations on the characteristics of the network traffic caused by connectionless RPC are summarized as follows:

   1. If the PDU size is larger than an IP packet, then the IP packets on the network come as PDU-related bursts, with interarrival times much more constant than is the case for inter-IP times. For some RPC implementations using single-IP PDUs, the network traffic caused by RPC is also bursty, but the bursts may be related to the characteristics of the RPC calls at the application level instead of the RPC PDUs.
2. The effects of interoperation are to delay the first IP packet (or PDU) of each RPC call, because of the longer marshalling time, and to delay the first PDU of the first time of communication, because of the negotiation on PDU size.

3. Increased network load causes contention when the network packets are sent, and increases the interarrival times of the transmission units.

4. There are many details to be taken care of in designing the connectionless RPC protocol, such as the length of the time-out period and the maximum size of the congestion window. When these parameters are improperly set, RPC may perform abnormally and badly.

Comparison of Protocols

The previous sections describe the measurement for the different RPC protocols separately, but there are obvious similarities: data fragmentation happens at both PDU and IP layers, and there are some similar characteristics for the network traffic caused by RPCs over these two RPC protocols. For example, when using multi-IP PDUs, the IP packets on the network come as PDU-related bursts, the intra-PDU-IP time is quite constant, while the inter-IP time is rather bursty.

The difference in the two protocols comes from the fact that they are built on top of different transport protocols (TCP vs. UDP), and flow control happens at different levels. Understanding the differences between the protocols, and the performance implications, helps in choosing the proper one when building the applications.

The interarrival time statistics from the two different RPC protocols are compared in Table 6.11 for the OS/2-to-OS/2 pair. This particular client/server pair is chosen because only for the OS/2-to-OS/2 pair, both connectionless and connection-oriented RPC protocols use multi-IP PDUs. The following observations are made:

1. Under connectionless RPC (RPC over UDP), both the mean inter-PDU time and the mean inter-IP time are larger than under connection-oriented RPC
(RPC over TCP). Only the mean intra-PDU-IP time is smaller. This shows that although PDUs come with larger spacing over connectionless RPC, the IP packets generated by a single PDU come with smaller spacing.

2. Under connectionless RPC (RPC over UDP), the CoVs of interarrival times for all transmission units are smaller. Moreover, the smaller CoV of intra-PDU-IP time shows that the IP packets comprising the same PDU have more constant interarrival times under connectionless RPC than under connection-oriented RPC.

Quite obviously, the network traffic is "burstier" under connection-oriented RPC, than it is under connectionless RPC, and the bursts of IP packets are even more PDU-related. This is true only when the PDU size is larger than one IP packet, so that both RPC and IP fragmentation happen.

Table 6.11: Comparison of Connectionless and Connection-oriented RPCs on Interarrival Time Statistics (OS/2-to-OS/2 Pair)

<table>
<thead>
<tr>
<th>Statistics</th>
<th>Transmission Unit</th>
<th>RPC over</th>
<th>Trace 1</th>
<th>Trace 2</th>
<th>Trace 3</th>
<th>Trace 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean PDU TCP</td>
<td>8.870</td>
<td>8.840</td>
<td>8.788</td>
<td>8.976</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interarrival IP UDP</td>
<td>3.906</td>
<td>3.982</td>
<td>3.914</td>
<td>3.942</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time (msec) IP-in-PDU TCP</td>
<td>1.341</td>
<td>1.322</td>
<td>1.304</td>
<td>1.354</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time (msec) IP-in-PDU UDP</td>
<td>1.157</td>
<td>1.211</td>
<td>1.150</td>
<td>1.152</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CoV of PDU TCP</td>
<td>0.726</td>
<td>0.729</td>
<td>0.780</td>
<td>0.795</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CoV of PDU UDP</td>
<td>0.567</td>
<td>0.613</td>
<td>0.593</td>
<td>0.598</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CoV of IP TCP</td>
<td>1.542</td>
<td>1.547</td>
<td>1.606</td>
<td>1.625</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CoV of IP UDP</td>
<td>1.325</td>
<td>1.368</td>
<td>1.358</td>
<td>1.365</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CoV of IP-in-PDU TCP</td>
<td>0.993</td>
<td>0.916</td>
<td>1.350</td>
<td>1.117</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CoV of IP-in-PDU UDP</td>
<td>0.483</td>
<td>1.031</td>
<td>0.464</td>
<td>0.463</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6.3 Measurement of RPC RTT and Throughput

End-to-end performance is also affected by the speed and capacity of the RPC communication between the DCE entities. Two metrics are used to measure the speed and capacity of RPC. They are round-trip time (RTT) and throughput, as described in Chapter 5. Network load, RPC protocols used, and interoperation are factors affecting the RTT and throughput when communicating with RPC.

6.3.1 RTT and Throughput of Connection-oriented RPC

Figure 6.10 shows the measurement of RPC RTT with different amounts of RPC data from client/server pairs of Digital UNIX-to-Digital UNIX, OS/2-to-OS/2, and Digital UNIX-to-OS/2, under different network loads. These plots show the average mean RTT from the traces taken for each client/server pair under a certain level of network load. Throughput results are shown in Figure 6.11. The plots are from the data of three traces of each client/server pair and different network loads. As when measuring the packet arrival process of RPC, the levels of network load are “light” and “heavy” for the OS/2-to-OS/2 pair, and “light” and “regular” for the Digital UNIX-to-Digital UNIX pair and the Digital UNIX-to-OS/2 pair.

For all three client/server pairs, the results show that the RTT increases with the RPC data amount, but there are some non-linear features. For example, RTT increases slowly with the RPC data amount when the data amount is small, and increases quite sharply when the RPC data amount is large.

RPC throughput decreases as the network load increases. This is easy to see in all three parts of Figure 6.11. Heavy network load, as shown in the OS/2-to-OS/2 pair result, decreases RPC throughput by more than 25%. In the Digital UNIX-to-Digital UNIX pair, the decrease is also obvious. Even in the Digital UNIX-to-OS/2 pair, where there was such a small number of users that the regular network load was only slightly heavier than the light network load, decreased RPC throughput is
also observed.

Increasing the network traffic causes the RPC RTT to increase. With more levels of network load, the relationship between the network load and RPC RTT can be seen more clearly. Figure 6.12 presents the measurement of the RTTs of RPCs containing different amounts of data under different levels of network load. From the plots, it can be seen that two factors caused the increase of RPC RTT: RPC data amount and network load. Obviously, the RPC data amount influences RPC RTT much more profoundly than the network load. Yet, it must be noted that the pattern of the added network traffic (introduced using the HP4972A LAN Protocol Analyzer) was not very bursty. In a “real” application environment with bursty network traffic from other sources, the effect of network load on the RTT, and also on the interarrival times discussed in the previous section, might be more pronounced.

The effect of interoperation on RPC performance is shown by the throughput measurement. RPC throughput is affected by the capacity of the communicating client and server machines. Comparing parts (a), (b), and (c) of Figure 6.11 shows that the Digital UNIX-to-Digital UNIX pair has the highest throughput, and the OS/2-to-OS/2 pair has the lowest throughput. This is not surprising since the Digital UNIX machine is bigger and faster. The Digital UNIX-to-OS/2 pair result can be attribute to changing the client machine in the OS/2-to-OS/2 pair to be a faster one, which caused RPC throughput to increase (compare part (c) with part (a)), or the result of changing the server machine of the Digital UNIX-to-Digital UNIX pair to be a slower OS/2 machine, decreasing RPC throughput (compare part (c) with part (b)). It can not be determined whether the client machine or the server machine affects RPC throughput more profoundly, lacking data from the OS/2-to-Digital UNIX pair because of the problem of DCE version in the experimental environment.
6.3.2 RTT and Throughput of Connectionless RPC

Figures 6.13 and 6.14 show the RPC round-trip time under different levels of network load, for different client/server pairs. The data from which the plots are generated is the average mean round-trip time across all available traces. Figure 6.15 shows RPC throughput results from the Digital UNIX-to-Digital UNIX pair and the Digital UNIX-to-OS/2 pair under different levels of network load.

As in the case of connection-oriented RPC, an increase in RPC RTT with increasing RPC data amount is observed for all the client/server pairs. Also, as happened in connection-oriented RPC, increased network load caused a decrease in RPC throughput, and an increase in RPC RTT, both of which are degradation of performance.

In the RPC RTT plot for the OS/2-to-OS/2 pair, there is a sharp jump between 4000 bytes and 6000 bytes. This jump is caused by the first RPC PDU. The PDU size for this client/server pair is 4312 bytes. When the RPC contains more data than that can be fitted into one PDU, it takes a longer time for the RPC to be transmitted because of the time required to generate and transmit those subsequent PDUs. Similar jumps can be found on the RTT plots of the Digital UNIX-to-Digital UNIX and the Digital UNIX-to-OS/2 pairs, except that the jumps appear between 1000 bytes and 1500 bytes, because the RPC PDU size here is 1464 bytes.

Interesting observations can be made from the plots of the Digital UNIX-to-OS/2 pair. There is an unexpected jump on the RTT plot (Figure 6.14 (b)) between 18000 bytes and 24000 bytes. The jump is oddly sharp and the increase in RTT is huge. This coincides with a long waiting period observed when the application was running, during which little, if any, network load was introduced between the client and the server. It appears that when the application client wanted to send a large amount of data (more than 18000 bytes) to the server, it was forced to wait. Also,
for the Digital UNIX-to-OS/2 pair, the RPC throughput is very low. Comparing
the left side plots of (a) and (b) in Figure 6.15 shows that under light network load,
changing the server machine from a Digital UNIX machine to an OS/2 machine
caused a 60% decrease in throughput, while in connection-oriented RPC, under
similar condition, the decrease in RPC throughput was only about 25%. All these
observations suggest that DCE RPC may behave improperly when interoperation
happens.

The sending window problem found in the measurement of interarrival times
suggests that further investigation of the window policy on sending and receiving
connectionless RPC PDUs is necessary. From the reply message from server to
client, the server-side receiving window size was read out to be 16 PDUs. This
means that the receiving window size is \(16 \times 1464 = 23424\) bytes. When sending
24000 bytes of data from the client to the server, the first 23424 bytes of data filled
up the receiving window, so that the last PDU containing the remaining 576 bytes
of data must wait for space in the receiving window. This delay for sending the wait­
ing PDUs not only increases the RPC RTT but also decreases the RPC throughput
significantly. The deduction is this kind of jump will appear whenever the RPC
data amount exceeds \(23424 \times n\) (where \(n\) is some integer \(\geq 1\)). The jump could
also happen on the RTT plot of OS/2-to-OS/2 pair, if the RPC data amount have
exceeded \(16 \times 4312 = 68992\) bytes. It is safe to use 16 as the receiving window size
in the OS/2-to-OS/2 pair, because RPCs are seldom used to transfer such a large
amount of data (68992 bytes) at one time. But when interoperation happens, and
the PDU size becomes smaller, the receiving window size becomes important, be­
cause it largely reduces the upper limit on the amount of data that one RPC call can
carry without tremendous performance degradation. This mismatching of receiving
window size and PDU size does suggest to the designers of RPC protocols that when
interoperation happens, adjusting only the PDU size may not be sufficient. At least
from the measurement results in this study, a negotiation on respective receiving
window size is also necessary when interoperating.
Now recall the large variance and mean values of inter-PDU (inter-IP, because the IP and PDU are identical here) time observed from the measurement of the packet arrival process of connectionless RPC. Obviously, the large variance is caused by the delay on the PDU containing the remaining 576 bytes of data. These PDUs make up only a small percentage of all PDUs, but because of the long waiting time of these PDUs, even the mean inter-PDU time is affected.

RPC round-trip times for connection-oriented and connectionless RPCs are compared in Figure 6.16, under light network load. Overall, connectionless RPC performs worse than connection-oriented RPC. The RTTs of NULL RPC for both RPC protocols are almost the same. When there is more data contained in the RPC call, connectionless RPC becomes slower. Under different RPC protocols for the Digital UNIX-to-OS/2 pair, the huge jump on the RTT plot is quite evident. The RTTs for connectionless RPCs containing less than 18000 bytes of data are roughly 10% larger than those for connection-oriented RPCs, but for data amounts larger than 18000 bytes, the RTT for connectionless RPC is about ten times that of connection-oriented RPC.

Previous measurement results in [42] indicated that for an OS/2-to-OS/2 client/server pair, connection-oriented RPC performed better (smaller round-trip times) when the RPC data amount was small, while connectionless RPC performed better when the RPC data amount was large. This was not the case in the experiments of this research. For the OS/2-to-OS/2 and Digital UNIX-to-OS/2 pairs, problems with the sending and receiving windows for connectionless RPC definitely affect performance. But even for the Digital UNIX-to-Digital UNIX pair, where there appear to be no windowing problems, connectionless RPC still performs worse. This gives the impression that in connectionless RPC there are more design decisions to make, and so people should be more careful when using and implementing connectionless RPC in DCE.

97
6.4 Summary

This chapter presented the measurement results from the experimental investigation of DCE RPC performance. The DCE RPC performance measurement had two goals: first, the packet-arrival process of DCE RPC-caused Ethernet traffic was measured in order to determine the characteristics of RPC workload; and second, the latency and capacity of DCE RPC was measured under different levels of three factors – RPC protocol, network load, and interoperation.

The measurement of the packet-arrival process in DCE RPC shows that for multi-IP PDUs, the DCE RPC-caused IP packets on the network tend to arrive in bursts. These bursts are PDU-related; that is, the spacings between those IP packets comprising one PDU tend to be smaller and more constant than the spacings between subsequent IP packets that belong to different PDUs, or the overall IP packet spacings. This observation is true regardless of network load level, DCE RPC protocols, and interoperation. For applications that use single-IP PDUs, the PDU-related bursts are less evident.

The DCE RPC RTT increases with the amount of data contained in the RPC call. Increasing network load also causes the RTT to increase, but the influence is not as pronounced as it is with the RPC data amount. This result is consistent with that reported in other studies. Finally, measuring RTT between different client/server pairs shows that interoperation with a slower machine causes RTT to increase.

Using connection-oriented RPC as opposed to connectionless RPC always results in lower values of RTT. Even for large RPC data amounts, connectionless RPC failed to perform better than connection-oriented RPC. This result held for different client/server pairs, even for the client/server pair where problems due to congestion window size and receiving window size were not found. This result con-
tradicts some previous research.

RPC throughput decreases when network load increases, and when interop­erating with a less capable machine. Connection-oriented RPC always yields higher throughput than connectionless RPC.

Internal factors have a significant impact on DCE RPC performance. Using connectionless RPC protocol requires more design decisions because it must handle the flow control at the PDU level. Because interoperation is a major requirement for DCE, and different platforms and different DCE implementations have different PDU sizes, the design of flow control is complicated. Failing to set the window size properly (as appears to happen in the OS/2 DCE version used in these experiments), or failing to negotiate proper sending and receiving window sizes between the client and server when interoperation happens (as appears to happen when the OS/2 and Digital UNIX machines interoperate), both resulted in unexpected performance degradation.

The abnormalities found in the experimental work reveal the importance of performance measurement to the design of computer systems. By studying these abnormalities, a better understanding of DCE RPC, especially the internal factors and design decisions has been obtained. These experiments also suggest areas for improvement in the current DCE RPC implementations. Designers of the DCE RPC protocol must consider the interoperation problem and take it into account, for example, in negotiations between different platforms on the sending and receiving windows of the flow control mechanism in connectionless RPC.

The next chapter deals with modelling the packet arrival process of DCE RPC-caused IP traffic on an Ethernet. The modelling work is based on the measurement results of this chapter, and provides further understanding of DCE RPC.
Figure 6.2: Interarrival Time Distributions of the Digital UNIX-to-Digital UNIX Pair Under Different Network Loads for Connection-oriented RPC
Figure 6.3: Interarrival Time Distributions of the Digital UNIX-to-OS/2 Pair Under Different Network Loads for Connection-oriented RPC
Figure 6.4: Mean Value and Standard Deviation of Inter-arrival Times for the OS/2-to-OS/2 Pair Under Different Network Loads for Connection-oriented RPC
Figure 6.5: Interarrival Times and Their Standard Deviations for Different Transmission Units under Light Network Load for Connection-oriented RPC.

Figure 6.6: Interarrival Times and Their Standard Deviations for Different Transmission Units under Light Network Load for Connectionless RPC, with the OS/2-to-OS/2 Pair.
Figure 6.7: Interarrival Time Distributions of the OS/2-to-OS/2 Pair under Light Network Load for Connectionless RPC
Figure 6.8: Interarrival Time Distributions for PDU and IP of the Digital UNIX-to-Digital UNIX Pair under Different Network Loads for Connectionless RPC

(a) Network Lightly Loaded

(b) Network Regularly Loaded

Figure 6.9: Interarrival Time Distributions for PDU and IP of the Digital UNIX-to-OS/2 Pair under Different Network Loads for Connectionless RPC

(a) Network Lightly Loaded

(b) Network Regularly Loaded
Figure 6.10: RPC Round-trip Time Measurement under Different Network Loads for Connection-oriented RPC
Figure 6.11: RPC Throughput Measurement under Different Network Loads for Connection-oriented RPC

Figure 6.12: The Effect of Different Network Loads on RPC RTT for Connection-oriented RPC
Figure 6.13: RPC Round-trip Time Measurement under Light Network Load for Connectionless RPC on the OS/2-to-OS/2 Pair

Figure 6.14: RPC Round-trip Time Measurement under Different Network Loads for Connectionless RPC on the Digital UNIX-to-Digital UNIX Pair and the Digital UNIX-to-OS/2 Pair
Figure 6.15: RPC Throughput Measurement under Different Network Loads for Connectionless RPC

(a) Digital UNIX-to-Digital UNIX
(b) Digital UNIX-to-OS/2

Figure 6.16: RPC Round-trip Time Measurement under Different RPC Protocols under Light Network Load

(a) OS/2-to-OS/2
(b) Digital UNIX-to-Digital UNIX
(c) Digital UNIX-to-OS/2
Chapter 7

Modelling the Packet Arrival Process of DCE RPC on Ethernet

7.1 Introduction

Modelling DCE RPC workload on an Ethernet consists of modelling the packet length and the packet arrival process. Since this research lacks real DCE applications, modelling packet length based on synthetically generated RPC data sizes makes little sense. This research concerns the modelling of the packet arrival process only. The measurement study described in the previous chapter shows that RPC-caused network traffic is bursty, and the bursts are related to the DCE RPC protocol processing. Simple models, such as a single-parameter Poisson model, may not be able to describe the bursts caused by multiple transmission layers.

Multi-level models have proven to be useful in describing other workloads [26]. Bursts of network packets relating to different levels of data fragmentation in file systems workloads, for example, undergo similar protocol processing to that of DCE RPC. The work to be described in this chapter is to study whether a simple model of the packet arrival process of DCE RPC workload on the Ethernet can represent real traffic as opposed to a more complicated model containing multiple levels.

Call-level modelling is not possible in this study, because both the data amount of an RPC call and the inter-call time are determined by user behaviour, which is
beyond the scope of this research. Instead, RPC-level and IP-level activities are the focus. The study is based on measurements from connection-oriented RPC, because both RPC and IP fragmentation happens for all client/server pairs for this protocol. Also, no problems were found in the flow control in the RPC implementation for this RPC protocol.

Section 1 describes previous studies of multi-level workload modelling. Section 2 discusses the models used in this characterization. Both models use measurement results from the previous chapter. Modelled interarrival times are compared with the measured values and distributions in Section 3.

7.2 Multi-level Packet Arrival Process Modelling

7.2.1 Workload Modelling of DCE RPC

Workload modelling is an important aspect of performance studies of computer systems. In DCE, RPC manages the communications between the clients and the servers, and RPC communication places a certain kind of workload on the network (which, in this research, is Ethernet). Workload modelling of DCE RPC on Ethernet provides understanding of both DCE RPC and the performance issues relating to network support for DCE systems and applications.

The basic requirement for a workload model is that it must be able to describe the characteristics of interest in the real workload of the system with acceptable accuracy. From the measurement results presented in the previous chapter, some important characteristics of the packet arrival process of the transmission units of DCE RPC Ethernet traffic are obtained. Other information, such as the length of the packets, is not available, since the measurement results come from synthetic applications. For this reason, the modelling work in this research deals only with
the packet arrival process of DCE RPC Ethernet traffic.

A good workload model also eliminates unnecessary details of the real workload. It balances simplicity and accuracy. Because in DCE the application client sends request RPCs to the server, and the server sends response RPCs to the client, they cause the same sort of traffic. It's sufficient for now to study one-way network traffic caused by the request RPCs.

### 7.2.2 Multi-level Arrival Process

The simplest way to model the workload of a system is to assume that the process by which events happen is Poisson, but this assumption has been challenged by many workload measurement and modelling studies (for example, [26], [27], and [47]). These measurements show that the ingredients of a system's workload, either packets on the network or requests to a server, arrive at the system as bursts with some degree of dependence among some of the workload ingredients. The burstiness is reflected in the coefficient of variation of the interarrival times of the packets or requests, which is much higher than one, the CoV value in the Poisson arrival model. These studies argue that simple models such as Poisson can provide only a few fundamental performance properties, and that more detailed models are needed for more comprehensive performance investigations.

Measurements show that the dependency of the ingredients (packets or requests) of a system's workload is caused by the multiple processes in generating the ingredients. For example, the Ethernet frames related to NFS (Network File System [45]) operations are generated by NFS packets (the NFS protocol units), and NFS packets are generated by NFS requests. Therefore the bursts of Ethernet frames depend on the arrival of NFS packets [26]. For another example, accesses to a file system come as "clusters". Within a single cluster of access, if a file is accessed it tends to be accessed again. Therefore the bursts of accesses to a file depend upon the clusters.
of access to a file system in which the file resides [47]. Both of these processes can be called *two-level arrival processes.*

From the measurement results in this research, dependency also exists in the arrival process of IP packets generated by DCE RPC requests because of PDU fragmentation and IP fragmentation. What happens with DCE RPC is quite similar to what happens with NFS, as described in [26]. The packet arrival process of DCE RPC-caused Ethernet traffic is also a multi-level arrival process. If the user behaviour (how and when the users make RPC calls) on the DCE client side is taken into account, there might be even more levels in the arrival process of DCE RPC-generated network packets.

### 7.2.3 Models for Multi-level Arrival Processes

Because of the existence of dependence, multi-level packet arrival processes cannot be modelled by simple models like Poisson. More complicated models need to be investigated. An easy way to do this is to simply use multi-level models, with each level of the model emulating the corresponding level in the multi-level arrival process. For example, modelling a two-level arrival process with a two-level model, the higher level in the model can be used to generate the arrival process of "clusters" of the ingredients, while the lower level of the model can be used to generate the arrival process of the ingredients within the "clusters", with the dependence of the two levels properly defined. The distributions or specific algorithms at each level of the model are decided by the model itself. This kind of model has been studied in some previous research.

A simple two-level model, the *packet tandem model*, was presented in [26]. The packet tandem model is based on the observation that data in a NFS request is first fragmented into NFS packets by the NFS protocol and then into IP packets by the IP protocol. As a result, the network traffic caused by NFS requests is made
up of "tandems" that are related to NFS packets. The IP packets comprising one
NFS packet are "cars" that make up the corresponding tandem. The maximum
inter-car time is smaller than the minimum inter-tandem time. The number of cars
contained in a tandem is referred to as tandem length, and is determined by the
amount of data in an NFS request and the size of the NFS protocol data unit. Ex­
periments in [26] compared the Packet Tandem Model and two other models that
take the arrival of packets as independent (Poisson model and histogram model),
and concluded that the packet tandem model captures the correlations between
the packet interarrival times and shows the burstiness characteristics of the workload.
These burstiness characteristics are not captured by the other two models, and can
significantly degrade Ethernet performance.

A more complicated multi-level model is the packet train model [27]. This model
concerns more details of the traffic and has more levels. In a packet train model,
the packet stream between two nodes consists of a number of "trains". The trains
consist of a group of "cars", where the cars are the network packets regardless of the
direction they travel (i.e. can originate from either node). Trains are identified by a
defined "maximum inter-car time". If the space between any two cars is larger than
this "maximum inter-car time", they are considered to be in two different trains.
This model describes the two-way workload on a network and has proven to be an
accurate (and subsequently very popular) way to describe such traffic.

Other workload models have used multiple level to generate bursts with depen­
dency. For example, [47] describes the two-level model and the algorithms of the
request arrival process of clients to a file server in a distributed file system.

Modelling multi-level arrival processes using multiple levels in the model has had
good results in previous workload studies and so this seems a reasonable approach
to take for this work. For example, the DCE RPC protocol and the NFS protocol
are similar in the fragmentation of application data: in both, application data is
first fragmented by the upper layer protocol (RPC or NFS protocol), and is further fragmented by the lower layer protocol (IP). This led to the hypothesis that a multi-level model can describe the packet arrival process of DCE RPC-caused Ethernet traffic better than a simple one-level model. To eliminate the effect of user behaviour on the generation of RPC calls, a two-level model is investigated. Experiments were done to compare the one-level and two-level structures of the packet arrival model using simple distributions of the workload characteristics provided by the measurements presented in Chapter 6.

7.3 The Models in the Experiment

Simple one-level and two-level models were used in the experiment to model the packet arrival process of the network traffic caused by DCE RPC. The parameters of the model are taken from the measurement results. These are the mean and variance of intra-PDU-IP time and the mean and variance of inter-PDU time. The packet arrival process of IP is modelled as mean inter-IP time, variance and coefficient of variation of inter-IP time, and full distribution of inter-IP times. Results from the models are then compared with measured values to show how accurate the models are, and which kind of model can better describe the packet arrival process of RPC introduced network packets. From all the traces, two were chosen at random to provide the parameters of the models and to be compared with the modelled results.

The primary one-level model is a simple Poisson model with one parameter. In a Poisson process, events (in this study, events are obviously the “arrival of the IP packets”) occur during non-overlapping intervals of time, and the occurrences of these events are independent. Also, the probability of one event occurring during a sufficiently small interval of time (Δt) is αΔt, where α is the event occurrence rate (1/α is the mean interarrival time of the events). The packet arrival process in the Poisson model is completely characterised by the mean value. The packets arrive according to an exponential distribution with the measured mean inter-IP time as a
parameter. An alternative one-level model is to use a hyper-exponential distribution to describe the IP packets' arrival. That is, instead of using only the mean value, the variance is also taken into account. These two one-level models are referred to as the one-level exponential (O.L.E.) model and the one-level hyper-exponential (O.L.H.) model, respectively.

In this experiment, the inter-IP times are generated by sampling from an appropriate exponential/hyper-exponential distribution, which takes the measured mean inter-IP time (and variances, for the later distribution) as the parameter. Table 7.1 shows the inter-IP times used in the O.L.E. model. Table 7.2 shows the inter-IP times used in the O.L.H. Model. The O.L.H. model is only applied to the packet arrival process under heavy network load, because only under heavy network load was the variance of inter-IP time big enough to be described by a hyper-exponential distribution.

Table 7.1: Parameters Used in the O.L.E. Model (from Measurement of Connection-oriented RPC, the OS/2-to-OS/2 Pair)

<table>
<thead>
<tr>
<th>Network Load</th>
<th>Mean Inter-IP Time of Sample Traces (msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Trace A</td>
</tr>
<tr>
<td>Light</td>
<td>3.498</td>
</tr>
<tr>
<td>Heavy</td>
<td>4.852</td>
</tr>
</tbody>
</table>

Table 7.2: Parameters Used in the O.L.H. Model (from Measurement of Connection-oriented RPC, the OS/2-to-OS/2 Pair)

<table>
<thead>
<tr>
<th>Inter-IP Time Statistics</th>
<th>Trace A</th>
<th>Trace B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean Value (msec)</td>
<td>4.852</td>
<td>5.161</td>
</tr>
<tr>
<td>Variance</td>
<td>116.232</td>
<td>131.381</td>
</tr>
</tbody>
</table>

Because of RPC fragmentation and IP fragmentation, the IP packets generated
by RPC data are not always independent. The measurements have shown that the IP packets come as PDU-related bursts, showing some degree of dependency. This motivates the modelling of the IP packet arrival process using a two-level model similar to the packet tandem model described in [26].

The two-level models used in this experiment are referred to as T.L.E.E., T.L.H.E., and T.L.H.H., as described in Table 7.3. The two levels of the model generate sampled inter-PDU times and sampled inter-IP times independently. One inter-PDU time is related to up to three inter-IP times. The number of IP packets comprising one PDU is determined by a value \( \delta \), which can be defined as the difference between the accumulation of sampled inter-IP times and the appropriate sampled inter-PDU time as follows:

\[
\delta = \text{sampled inter PDU time} - \sum_{i=1}^{n} \text{sampled inter IP time}_i
\]

Here \( 1 \leq n \leq 3 \), because the PDU length is 3 IP packets in the protocol. Dependency happens when deciding how many IP packets are in one PDU and the inter-IP time between the last IP in a PDU and the first IP in the next PDU. Because the dependency is caused by PDUs, the inter-PDU time must also be used in the calculation. Figure 7.1 shows how the dependency is taken into account according to the \( \delta \) and \( n \) values.

(A) \( n = 3 \), and \( \delta \geq 0 \): 3 IP packets are generated, and the modelled inter-IP time between the third IP packet to the first IP in the next PDU is calculated by:

\[
\text{sampled inter PDU time} - \sum_{i=1}^{3} \text{sampled inter PDU IP time}_i
\]

(B) \( n = 3 \), and \( \delta < 0 \): 3 IP packets are generated and contained in this PDU. The modelled inter-IP time between the third IP in this PDU and the first IP in the next PDU is calculated by:

\[
\text{sampled inter PDU time} - \sum_{i=1}^{2} \text{sampled inter PDU IP time}_i
\]

(C) \( n = 2 \), and \( \delta \leq 0 \): 2 IP packets are generated and contained in this PDU. The
modelled inter-IP time between the second IP in this PDU and the first IP in the next PDU is calculated by:

\[ \text{sampled inter-PDU time} - \text{sampled inter-PDUIP time}_1 \]

(D) \( n = 1 \), and \( \delta \leq 0 \): This PDU contains only one IP packet. The sampled inter-PDU time is the modelled interarrival time between this IP packet and the first IP packet in the next PDU (inter-IP time).

Table 7.3: The Two-level Models

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Upper Level Distribution</td>
<td>Exponential</td>
<td>Hyper-exponential</td>
<td>Hyper-exponential</td>
</tr>
<tr>
<td>Lower Level Distribution</td>
<td>Exponential</td>
<td>Exponential</td>
<td>Hyper-exponential</td>
</tr>
</tbody>
</table>

Tables 7.4 and 7.5 show the parameter values used in the two-level models. These values are derived from the measurement results presented in Chapter 6. The upper level of the model uses measured inter-PDU times, and the lower level uses the measured intra-PDU-IP times. Characteristics of the modelled inter-IP times (mean values, variances, and full distribution) are then compared with the measured inter-IP times to determine whether or not the model is able to describe the network workload characteristics satisfactorily. This comparison is presented in the next section.

7.4 Analysis of the Modelling Results

Results are shown and compared in Table 7.6 for light network load. Under light network load, only the exponential distribution is used because the variance is not big enough to fit into the range of hyper-exponential. Comparing the modelled inter-IP times with the measured ones, the modelled mean inter-IP time from the
Figure 7.1: The Two-level Models in the Experiment

Table 7.4: Parameter Values (Measured Interarrival Time Statistics) Used in the T.L.E.E. Model

<table>
<thead>
<tr>
<th>Network Load</th>
<th>Transmission Unit</th>
<th>Mean (msec) Trace A</th>
<th>Mean (msec) Trace B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Light</td>
<td>PDU (level 1)</td>
<td>8.870</td>
<td>8.840</td>
</tr>
<tr>
<td></td>
<td>PDU-IP (level 2)</td>
<td>1.341</td>
<td>1.322</td>
</tr>
<tr>
<td>Heavy</td>
<td>PDU (level 1)</td>
<td>14.973</td>
<td>15.657</td>
</tr>
<tr>
<td></td>
<td>PDU-IP (level 2)</td>
<td>2.551</td>
<td>2.735</td>
</tr>
</tbody>
</table>

O.L.E. model (ie. Poisson model) is quite good. This is not surprising because the measured mean inter-IP time is the parameter of the model. But, the agreement of variance and CoV values is not as good. CoV values are almost 1.0, showing no bursts, which is far from what happened on the network. On the other hand, al-
Table 7.5: Parameter Values (Measured Interarrival Time Statistics) Used in the T.L.H.E. Model and T.L.H.H Model - Heavy Network Load Only

<table>
<thead>
<tr>
<th>Transmission</th>
<th>Trace A</th>
<th>Trace B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean (msec)</td>
<td>Variance</td>
</tr>
<tr>
<td>PDU (level 1)</td>
<td>14.973</td>
<td>375.952</td>
</tr>
<tr>
<td>PDU-IP (level 2)</td>
<td>2.551</td>
<td>52.536</td>
</tr>
</tbody>
</table>

though the modelled mean inter-IP times from the T.L.E.E. model is smaller to the measured values, the variances are much closer to the measured variances, and the CoVs show the characteristics of burstiness. The full distributions of inter-IP times, both modelled and measured, are shown in Figure 7.2. The distribution of measured inter-IP times is quite concentrated, with a long tail, while the O.L.E. model results in a much less concentrated inter-IP time distribution and with a much shorter tail. The distribution of inter-IP times from the T.L.E.E. model is more concentrated, and the tail is longer than that from the O.L.E. model.

Table 7.6: Comparing Modelled and Measured Values for Inter-IP Process Under Light Network Load over Connection-oriented RPC, the OS/2-to-OS/2 Pair

<table>
<thead>
<tr>
<th>Source of Values</th>
<th>Trace A</th>
<th>Trace B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Var</td>
</tr>
<tr>
<td>T.L.E.E.</td>
<td>2.656</td>
<td>23.113</td>
</tr>
<tr>
<td>Measurement</td>
<td>3.498</td>
<td>29.103</td>
</tr>
</tbody>
</table>

In reality, the network is not always lightly loaded. From the measurement results in Chapter 6, it was determined that the network load affects the arrival pattern of IP packets. DCE RPC-caused IP traffic was also studied on a heavily loaded network using one- and two-level models. Table 7.7 compares the modelled and measured inter-IP times. Five models are involved: the one-level models are the O.L.E. model (ie. the Poisson model) and the O.L.H. model; the two-level models
Figure 7.2: The Modelled and Measured Inter-IP Time Distribution Under light Network Load for Connection-oriented RPC, the OS/2-to-OS/2 Pair

Table 7.7: Comparing Modelled and Measured Values for Inter-IP Process Under Heavy Network Load over Connection-oriented RPC, the OS/2-to-OS/2 Pair

<table>
<thead>
<tr>
<th>Source of Trace</th>
<th>Trace A</th>
<th>Trace B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Results</td>
<td>Mean</td>
<td>Var</td>
</tr>
<tr>
<td>O.L.E.</td>
<td>4.857</td>
<td>23.884</td>
</tr>
<tr>
<td>O.L.H.</td>
<td>4.799</td>
<td>112.103</td>
</tr>
<tr>
<td>T.L.E.E.</td>
<td>4.628</td>
<td>63.996</td>
</tr>
<tr>
<td>T.L.H.E.</td>
<td>4.824</td>
<td>110.012</td>
</tr>
<tr>
<td>T.L.H.H.</td>
<td>5.045</td>
<td>143.389</td>
</tr>
<tr>
<td>Measurement</td>
<td>4.852</td>
<td>116.232</td>
</tr>
</tbody>
</table>
are the T.L.E.E. model, the T.L.H.E. model, and the T.L.H.H. model.

The mean inter-IP time is best modelled by the O.L.E. model. The T.L.H.E. model and the O.L.H. model are almost equally good at modelling the mean inter-IP time. The modelled mean inter-IP times from other models are similar, except that the T.L.H.H. values are a bit too high. Big differences are also seen in the variances and CoVs. The closest variances are from the O.L.H. model and the T.L.H.E. model. The O.L.E. model results in small variances and the CoVs are near 1.0. The O.L.H. model models the variance well, because the measured variance is part of the input. T.L.E.E. results in smaller variances and T.L.H.H. results in higher variances. It appears that these two models either underestimated (T.L.E.E) or overestimated (T.L.H.H) the burstiness of PDU and PDU-IP traffic. Nevertheless, the CoV values from these two models match measured values better than do those from the O.L.E. model. Overall, the T.L.H.E and O.L.H models appear to provide close agreement with measured values. Details of the modelled and measured inter-IP times are shown in Figure 7.3.

For the heavily loaded network, using the same distribution (eg. exponential distribution or hyper-exponential distribution), the two level models describe the packet arrival process of DCE RPC-caused Ethernet packets better. This is shown by the full distributions, variances and coefficients of variation. The mean inter-arrival time is easy to model, but can not show the bursty characteristics of the traffic. For the two-level models, the T.L.H.E. model works best from perspective of summary statistics. With respect to full distributions, the plot from the T.L.H.H. model has a similar shape to the measurement plot, but obviously is shifting left with a longer tail. This means that when assuming that the intra-PDU-IP times are hyper-exponentially distributed, the bursty nature of the traffic is over-estimated. For the fractions of inter-IP times that are less than 3 msec, the inter-IP times generated by the T.L.E.E. and the T.L.H.E. models are much nearer to the measurement results. Comparing the tail lengths of the plot from the two models, the
Figure 7.3: The Modelled and Measured Inter-IP Time Distribution Under Heavy Network Load for Connection-oriented RPC, the OS/2-to-OS/2 Pair

plot from the T.L.H.E. model has a longer tail, and results in larger variances than the T.L.E.E. model. The reason that T.L.H.E. model works best is that it considers
the inter-PDU time to be more bursty than the intra-PDU-IP times, and this fits the observations from measurement. On the other hand, the success of the T.L.H.E. model also supports the conclusion from measurement for the RPC-caused IP packet arrival process on Ethernet: RPC-caused IP traffic is bursty, with bursts related to the arrival of PDUs. The arrival of most IP packets comprising one PDU is evenly spaced.

To summarize, for both lightly and heavily loaded networks, two-level models appear to work better than one-level models. The modelled mean inter-IP times from the one level models are nearer to the measured mean inter-IP times than are the mean inter-IP times modelled by the two-level models. This does not mean that one-level models are better. Considering the deviations of the variance and the coefficients of variation from the measured values, the model needs to make changes to the parameters in order to be properly calibrated. But changing the input parameter (the mean and variance of the inter-IP times) certainly causes the modelled values of mean inter-IP time (and variance for the O.L.H. model) to deviate from the measured value. This means the "good" modelled mean inter-IP times (and variances in the O.L.H. model) are actually a drawback for this kind of model.

For different two-level models, the distribution chosen for each level determines the quality of the model. The distribution selected to sample the interarrival times on a certain level should be able to describe the characteristics of the interarrival times of this level.

7.5 Summary

The measurement of packet arrival process of DCE RPC-generated IP packets on Ethernet showed that the traffic is bursty. The bursts of IP packets on the network are related to the appearance of DCE RPC PDUs. This suggests that a simple single-level model can not describe the IP packet arrival process precisely.
To study possible structures for models (particularly levels), five simple models are investigated in this chapter: O.L.E. model (Poisson model), O.L.H. model, T.L.E.E. model, T.L.H.E. model, and T.L.H.H. model. Measured mean inter-IP time and variance of inter-IP time are used as the parameter values for the one-level models. Measured mean and variance of inter-PDU time, and mean and variance of inter-PDU time are used as the parameter values for the two-level models.

Comparing the modelled and measured inter-IP times on both statistical values and full distributions shows that one-level models are not able to describe the bursty characteristics of the DCE RPC-caused network traffic as well as the two-level models. Among the two-level models, because the T.L.H.E. model captured the characteristics that inter-PDU times are variant when the network load is heavy while the interarrival times of IP packets comprising one PDU is quite uniform, the T.L.H.E. model appears to be the best among the three models in modelling the packet arrival process of IP packets generated by DCE RPC on the network.
Chapter 8

Summary and Future Work

This chapter summarises the research described in this thesis. The first section reviews the goals of the research, and the achievements with respect to these goals. This is followed by a section discussing what needs to be done in the future.

8.1 Thesis Summary

Distributed computing is popular in today's computer world. The development of distributed applications is simplified by using the services provided by middleware systems. As the de facto standard of middleware, OSF's DCE was chosen as the focus of this research to shed light on performance issues concerning middleware and the distributed applications running on middleware systems.

Communication is a core issue in the performance of distributed applications and middleware systems. The quality of communication between the nodes running distributed applications and middlewares heavily influences the performance of both the distributed applications and the middleware which supports the applications. In DCE, communications are based on the Remote Procedure Call (RPC) protocol.

The main goal of this study is to understand DCE RPC from the performance perspective. This goal leads to three objectives in the study:

1. Understanding the mechanisms and design decisions of DCE RPC. This pro-
vides insight into DCE RPC, and helps in determining the internal and external factors that affect RPC performance in a DCE environment.

2. Measuring the performance of DCE RPC. This includes the measurement of RPC latency and capacity under different levels of different factors, and the measurement of the packet arrival process of DCE RPC-caused network traffic. This shows how DCE RPC performs, and the characteristics of DCE RPC-caused workload.

3. Modelling the packet arrival process of DCE RPC-caused traffic on the network. In doing so, more understanding of DCE RPC and its workload on the network can be obtained.

The study of the mechanisms and design decisions of DCE RPC was described in Chapter 2 and 3. In Chapter 2, the role DCE RPC plays in DCE and DCE applications was discussed, along with a background information relating to the structure of distributed applications in DCE. In Chapter 3, details of DCE RPC, such as the runtime processing and the RPC protocol, were described.

A distributed application in DCE is specified by the application interface and consists of the interface definition, client procedure, server procedure, and the remote procedure on the server side. Details of the client/server communication are not handled by the application, but by client and server stubs. Stubs are generated by the DCE RPC mechanism according to the interface definition. The DCE RPC Runtime supports the running of stubs according to the RPC protocols, and manages the client/server communications. With this structure, application developers need not concern themselves with the details of communication, and therefore, application development is simplified. DCE RPC calls and the data contained in the calls are sent from the application to the RPC runtime, and from one layer of protocol to another until they reach the network. This process adds to both the amount of data to be transferred and the time to process a DCE RPC call. Its existence
certainly affects DCE RPC performance.

Fragmentation is a process of cutting data into pieces whose size is determined by the protocol, and adding headers to these data pieces to form a protocol data unit (PDU) or packet. Fragmentation happens on every protocol layer. In DCE RPC communication, RPC fragmentation, transport layer (TCP or UDP in this research) fragmentation, and IP fragmentation all occur. The size of the DCE RPC PDU is an important internal factor in DCE RPC performance because it determines whether or not data cutting happens in lower protocol layers, and the RPC PDU is the unit upon which flow control works for connectionless RPC. Another important internal factor is the flow control strategy used within DCE for connectionless RPC. Investigation of the flow control strategy of DCE RPC shows that the send and receive window (or buffer) sizes affect the time a packet is put onto the network. Different types of DCE RPC protocol (connection-oriented and connectionless) have different sizes of RPC PDU, and determine on which layer flow control happens. Therefore the type of RPC protocol used within a DCE application is an important external factor in DCE RPC performance. Different DCE RPC implementations on different operating systems and platforms have different RPC PDU sizes, and sometimes different flow control buffer sizes, and so this becomes an external factor affecting DCE RPC performance. The load on the network is another external factor.

Most previous DCE RPC performance studies considered the measurement of DCE RPC round trip time (RTT) and throughput. They found that DCE RPC RTT is heavily affected by the amount of data contained in the DCE RPC call. Depending on different experimental environments, the increase of RPC RTT with the amount of RPC call data is either linear or none linear. Within a limited amount of RPC data (not greater than 20K bytes), the increases observed from different environments are similar. Some of the research showed that the RTTs of connectionless RPC are smaller than those of connection-oriented RPC when the RPC data amount exceeds a certain limit (eg. 4K bytes). Increasing network load degrades
RPC performance. These results were reviewed in Chapter 4.

This research was carried out in the DISCUS Lab of the Department of Computer Science in the University of Saskatchewan. The experimental DCE cell contains four nodes of different platforms and operating systems working as DCE application clients and servers, a network monitoring station and a measurement station. All these computers are connected by a 10 Mbps Ethernet local area network. A synthetic DCE application was used to generate the RPC workload. The goal of the measurement was two fold: to measure DCE RPC speed and capacity, and to characterise the packet arrival process of DCE RPC-caused Ethernet traffic. The description of the environment and the experiments, the definitions of the performance metrics, and the levels of the performance factors make up the content of Chapter 5.

The measurement results and analysis of the results are presented in Chapter 6. Measurement of RPC RTT and throughput in the DCE environment shows that the increasing of RTT with the RPC call data amount is similar to the results of previous research. Jumps on the plot of RTT vs. data amount were observed where the first PDU or IP is formed. An increase in network load causes an increase in RTT, but the impact of this factor is much less than that of the RPC data amount. The increase of network load increases RTT and also decreases the data throughput (bytes per second) of DCE RPC. Interoperating RPC between different platforms results in performance better than the "slower" machine pair, and worse than the "faster" machine pair.

Statistics of interarrival times of DCE RPC transmission units for multi-IP PDU implementations show that mean intra-PDU-IP time is the smallest, and mean inter-PDU time is the largest. The variance of the inter-IP times is the largest and the variance of the intra-PDU-IP times is the smallest. The coefficient of variation of intra-PDU-IP times is always less than 1.0. All these results indicate that for RPC protocols with multi-IP PDUs, the IP packets on the network caused by the DCE
RPC application always come as PDU-related bursts.

A measurement result that is at odds with previous RPC performance measurements was found in the RPC RTT measurement between different RPC protocols. The measurement in this research show that connectionless RPC, whatever amount of data it contains, performs worse than connection-oriented RPC.

An unexpected second peak on the inter-PDU time distribution plot for connectionless RPC on OS/2 client/server pairs suggests a problem in the way the congestion window is expanded on the application client side. A “bug” or design error might set the client side congestion window size to be 1, making all subsequent PDUs from the same RPC call to wait for the acknowledgment of its previous PDU. This could be a source of significant performance degradation for DCE RPC in OS/2 environments.

For connectionless RPC, another large increase in RTT was observed for the Digital UNIX-to-OS/2 pair when the data was more than 24K bytes. This is thought to be caused by mismatching sending and receiving window sizes. When interoperation happened, the client and server negotiated a smaller PDU size, but stuck to their original sending/receiving window sizes. This caused the maximum amount of data that can be held in the receiving window to decrease, and resulted in performance degradation when a large amount of data was sent. This observation suggests that more issues need to be taken into account when interoperation happens; for example, negotiation on sending and receiving window sizes based on the PDU size.

From the observations in this research, it seems that connectionless RPC has more design decisions to make, and performs worse than connection-oriented RPC. It also needs more work on negotiation when interoperation happens, and therefore should be avoided when writing DCE applications.
The modelling experiment shows that two-level models which consider the dependencies between the sub-sequential IP packets generated by the same RPC PDU were more effective than the one-level models in describing the burstiness of the network traffic caused by DCE RPC applications. The one-level models are good at modelling the mean value of interarrival times, but can not capture the bursty and dependent nature of the workload described by the variance and coefficient of variation. For a lightly loaded network, the T.L.E.E. model best describes the IP packet arrival process on Ethernet when running DCE RPC applications, while for a heavily loaded network, the T.L.H.E. model is best. The modelling work is presented in Chapter 7.

8.2 Future Work

The measurement and modelling work of this research has shed more light on DCE RPC – in particular, with respect to the organisation of DCE RPC applications, the DCE RPC protocols, and the data processing when it is transmitted. By measuring DCE RPC performance when running synthetic applications, better understanding of DCE RPC performance issues has been obtained. The characterisation of the packet arrival process of DCE RPC-caused network traffic and the subsequent modelling of it have provided a good start at modelling of the workload of DCE RPC.

Based on the experience gained in this study, future work on DCE consists of two parts: further performance measurement, and in-depth workload study. More performance measurement is need because of the limitation of the current experimental environment: lack of real DCE applications, mismatching DCE versions on different platforms that caused communication failure in certain client/server pairs, and implementation problems in the Beta version running on OS/2 machine all created problems. The workload study in this research concerns only the packet arrival process. A more complete workload model needs to be built in the future. The two
parts of the future work are discussed further in the following paragraphs.

When there are real DCE applications running in the environment, the measurement of DCE RPC workload on Ethernet can be expanded to include measurements of factors such as packet length and packet data size. With real DCE applications, the call length, and inter-call times are also worth measuring, giving more information about the DCE RPC-caused workload on the network.

Interoperation is a major goal of DCE, and a big concern in this research. Yet the study on interoperation is far from complete because there were different DCE versions running on the different platforms. The client/server communication failed when using the Digital UNIX machine (which was running a lower version of DCE) as server and the OS/2 machine (which was running a higher version of DCE) as client. Lacking this part of the experiment prevents us from knowing whether the client or the server machine affects RPC performance more profoundly. More study is also needed on window size negotiation when interoperation happens over connectionless RPC. Simulation might be useful in this direction. Also, there are other interesting issues concerning interoperation that would be worth studying in the future.

In the modelling work the two-level model has been shown to describe the packet arrival process better than a one-level model. With the information gained in the new measurement, a more complete model such as the Packet Tandem Model [26] could be developed. It is also possible that the workload characterisation shows that more levels in the model is needed, and leads to a more complicated model. For example, with real DCE applications, call-level measurement could provide information such as call length and inter-call time, and could well introduce a new level in the model. It seems clear that any future workload model for DCE RPC on Ethernet should be multi-leveled, with dependency between different levels. More details such as packet length, packet data size and number of levels are needed for
the development of future workload models.

This research has helped to shed light on the role and working mechanism of RPC in DCE, has done some basic measurements of DCE RPC performance, and has provided a prototype for DCE RPC workload study. It is believed that studies of RPC mechanisms and performance issues can contribute a lot to the improvement of DCE and middleware performance. Further studies in this direction will bring more understanding and performance improvement to DCE, RPC, middleware, and distributed applications.
References


